

Algorithm Engineering for Hard Problems in Computational Geometry

Der
Carl-Friedrich-Gauß-Fakultät
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung des Grades eines *Doktors der Naturwissenschaften (Dr. rer. nat.)*

vorgelegte Dissertation

von *Dominik Michael Krupke*

geboren am 23.04.1992

in Göttingen.

Eingereicht am: 11.01.2022

Disputation am: 07.03.2022

1. Referent: Prof. Dr. Sándor P. Fekete

2. Referent: Prof. Dr. William J. Cook

3. Referent: Prof. Dr. Joseph S. B. Mitchell

2022

Dissertation

Colophon

This document was typeset with the help of KOMA-Script and L^AT_EX using the kaobook class.

It's like everyone tells a story about themselves inside their own head.
Always. All the time. That story makes you what you are. We build
ourselves out of that story.

– Patrick Rothfuss, *The Name of the Wind*

Abstract

Many practically relevant problems in Computational Geometry are provably hard to solve. In this thesis, we consider a series of these problems and tackle them with techniques from algorithm engineering, e.g., mixed integer programming or deep reinforcement learning. Additionally, we provide a number of results on complexity and algorithms for these problems.

We start with a set of problems occurring in the context of large satellite systems, where communication requires a rotation of directional antennas. Here, we discover a close relationship to the vertex coloring problem, use constraint programming to obtain optimal solutions, and provide a practical auction-based algorithm that achieves good results in a realistic simulation. Then we continue with (partial) coverage path planning problems involving turn costs, which require different approaches than when minimizing only the traveled distance. We engineer an approximation algorithm based on linear relaxation and minimum-weight perfect matching to solve large instances in grid graphs. This approach is then generalized to complex polygonal instances, which allow modeling of difficult real-world scenarios. Afterwards, we focus on probing a set of trajectories to maximize the captured information, where we successfully apply mixed integer programming and a set of heuristics. We also consider robots so small they need to be actuated by external forces like a magnetic field; applications for such robots include cancer treatment. Construction plans for miniature objects are computed using SAT-solvers, and control sequences to gather a swarm of these robots are optimized using deep reinforcement learning. Finally, we close the thesis with an excursion in hosting the CG:SHOP competition for three years and counting.

Zusammenfassung

Viele der praktisch relevanten Probleme in der Computational Geometry sind beweisbar schwer zu lösen. In dieser Ausarbeitung betrachten wir eine Reihe solcher Probleme und lösen sie mit Techniken aus dem Algorithm Engineering, z.B., Mixed Integer Programming oder Deep Reinforcement Learning. Zusätzlich erlangen wir etliche Erkenntnisse zur Komplexität und Algorithmen für diese Probleme.

Wir beginnen mit diversen Problemen, die im Kontext von größeren Satellitensystemen auftreten und bei denen die Kommunikation das Rotieren von gerichteten Antennen erfordert. Hier entdecken wir einen engen Zusammenhang zum Graphenfärbungsproblem, nutzen Constraint Programming zur Berechnung von optimalen Lösungen und entwickeln einen praktischen, auktionsbasierten Algorithmus, der gute Ergebnisse in realistischen Simulationen erzielt. Danach fahren wir mit dem (Partial) Coverage Path Planning Problem unter der Berücksichtigung von Abbiegekosten fort, welches andere Ansätze braucht, als wenn wir nur die gefahrene Distanz minimieren wollen. Hierfür implementieren wir einen Approximationsalgorithmus — basierend auf linearer Relaxierung und Minimum-Weight Perfect Matching — auf eine Art, die es ermöglicht, auch große Instanzen im Gittergraphen zu optimieren. Dieser Ansatz wird anschließend von uns für komplexe polygonale Instanzen erweitert, was die Modellierung von schwierigen, realistischen Szenarien erlaubt. Anschließend fokussieren wir uns auf die Maximierung von eingeschlossenen Informationen aus einer Menge von Trajektorien, wo wir erfolgreich Mixed Integer Programming sowie eine Menge von Heuristiken anwenden. Wir betrachten auch Roboter, die so klein sind, dass sie nur durch äußere Kräfte — wie etwa ein magnetisches Feld — bewegt werden können und einen potentiellen Einsatz in der Tumorbehandlung haben. Konstruktionspläne für Miniaturobjekte berechnen wir mittels SAT-Solovern, und Kontrollsequenzen zum Sammeln eines solchen Roboterschwarms optimieren wir mittels Deep Reinforcement Learning. Letztlich schließen wir diese Arbeit mit einem Ausflug in die Organisation der CG:SHOP Challenges.

Preface

This thesis is a result (but not the end) of over five years of research, over ten years of study, and the collaboration with many great minds. I tried not only to capture my research results, but also to share the techniques and insights I learned over this time and deemed useful. In this thesis, I disclose most of my portfolio of techniques by applying it to various interesting optimization problems, and it is my desire that each reader will be able to find a useful takeaway for their own portfolio. If you are not looking to increase your portfolio but are interested in the considered optimization problems, I highlighted directions for future work in multiple places as described below.

Be aware that the content of this thesis is not ordered chronologically, rather by topic, and many chapters have already been published as individual papers. Some sections may be significantly older or newer than the surrounding content, which can lead to slight inconsistencies in the application of techniques and their presentations.

Future Work Throughout this thesis, there are many blue boxes containing suggestions for future work, illustrated below. These boxes are intended as starting point and inspiration for anyone interested in joining this field of research. Some of the questions are difficult and some just did not fit into the scope of this thesis. The conclusions of the individual chapters may also contain further suggestions for future work, if they did not fit directly into the text.

Future Work 0.1.

This is an example for a box describing a question for future work.

Plots True to the spirit of algorithm engineering, we will encounter many plots throughout this thesis. Let us quickly recap how to interpret the different types of plots. The most common plot is the line plot, with its lines showing the mean value, and the transparent extensions showing the 95 % confidence interval. The wider the confidence interval, the higher the deviation of the data. For computing the mean and the confidence interval, the x-axis is usually aggregated in some way, e.g., by incorporating values within a $\pm 5\%$ range. Most experimental data has too strong a deviation to be representable otherwise. This type of plot is very useful to present a value, e.g., the runtime, dependent on a parameter, e.g., the instance size. If we need to compare more than a few different configurations, line plots become too crowded, so we switch to bar plots or box plots. Bar plots only show the mean, while box plots show the mean, quartiles, range, and outliers. Outliers lie outwith the inter-quartile range. If the first quartile ends at q_1 and the third quartile at q_3 , then points smaller than $q_1 - 1.5 \cdot (q_3 - q_1)$ or larger than $q_3 + 1.5 \cdot (q_3 - q_1)$ are defined as outliers. Scatter plots are raw data without any aggregation. The last and most complex type of plot is the pair plot, which consists of a set of scatter plots along with density distributions. These are used to show the correlation between multiple features and the value distribution within the features. You can easily find more information on how to read such plots online using the corresponding keywords. Most plots and figures are unfortunately not optimized for black and white prints or colorblind people.

Supplementary material Most of the code and data is available on <https://github.com/d-krupke/dissertation>.

Acknowledgement This thesis would not been possible without a number of collaborators and supporters. First and foremost, I thank my PhD-advisor Sándor Fekete, who supported me since the very early stages, years before I started my dissertation. His support was unwavering, even when I failed. Next, I would like to thank the current and prior members of Sándor's research group: Andreas Haas, Phillip Keldenich, Linda

Kleist, Matthias Konitzny, Christian Rieck, Christian Scheffer, and Arne Schmidt. For each and every person, I can point to sections in this thesis that would have been impossible in their current form without their assistance. Additionally, they voluntarily reduced my workload during the last months to help me finish this thesis (especially Phillip). Of course, Ute Marchot has to be mentioned too: she not only alleviated my suffering from tedious paperwork, but she also proved to be someone I can count on. Of my students, Alexander Hill significantly contributed to two chapters, but also Michael Perk and Jonas Dippel added valuable content. Further, I would like to list the external collaborators who contributed to this thesis: Aaron Becker, Kevin Buchin, Erik Demaine, Li Huang, Irina Kostitsyna, Roel Lambers, Sheryl Manzoor, Tyler Mayer, Joe Mitchell, Ojas Parekh, Cynthia Phillips, and Martijn Struijs. The following ASIMOV project members should be acknowledged as they created the foundation for the part on satellites: Mohamed Khalil Ben-Larbi, Mirue Choi, Benjamin Grzesik, Tom Haylok, Harald Konstanski, Kattia Flores Pozo, Volker Schaus, Christian Schurig, and Enrico Stoll. Monica Wagner not only support me emotionally, but also greatly assisted in the grammatical fine-tuning of this thesis. Further proofreading was performed by Maximilian Ernestus, Aaron Becker, Phillip Keldenich, Matthias Konitzny, Johanna Krupke, Linda Kleist, Victoria Sack, and Marco Nikander. My other friends and family deserve acknowledgement for keeping me (in)sane over all these years.

This work was partially funded by DFG project “Computational Geometry: Solving Hard Optimization Problems” (CG:SHOP), FE 407/21-1.

Dedication This dissertation is dedicated to my beloved grandfather, Otto Krupke, who taught me to work hard and be diligent, but also to enjoy life. He passed away during the final stage of this thesis, after being my strongest supporter and role model since I can remember. I hope this thesis would have made him proud.

Disclosure

Around two-thirds of the material presented in this thesis has already been published in proceedings and journals, and arose from collaborations with various coauthors. The following list provides an overview of all previously published material and the corresponding collaborations. An additional disclosure can be found as a footnote at the beginning of each corresponding chapter. The authors in the following papers (except for 4, where I was first author, and 8, where only one particular result is used) are listed independent of their contributions in strictly alphabetical order, following the conventions in the corresponding fields. The papers 1, 3, 4, and 5 were personally presented at the corresponding conferences, and paper 10 at a preceding workshop. According to the character of a thesis, only content with substantial personal contributions has been included in this dissertation and may have been significantly altered, i.e., shortened, extended, or improved. All further content is novel and independent research.

1. Sándor P. Fekete, Linda Kleist, and Dominik Krupke. 'Minimum Scan Cover with Angular Transition Costs'. In: *36th International Symposium on Computational Geometry, SoCG 2020, June 23-26, 2020, Zürich, Switzerland*. Ed. by Sergio Cabello and Danny Z. Chen. Vol. 164. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020
2. Sándor P. Fekete, Linda Kleist, and Dominik Krupke. 'Minimum Scan Cover with Angular Transition Costs'. In: *SIAM J. Discret. Math.* 35.2 (2021)
3. Kevin Buchin, Sándor P. Fekete, Alexander Hill, Linda Kleist, Irina Kostitsyna, Dominik Krupke, Roel Lambers, and Martijn Struijs. 'Minimum Scan Cover and Variants - Theory and Experiments'. In: *19th International Symposium on Experimental Algorithms, SEA 2021, June 7-9, 2021, Nice, France*. Ed. by David Coudert and Emanuele Natale. Vol. 190. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021
4. Dominik Krupke, Volker Schaus, Andreas Haas, Michael Perk, Jonas Dippel, Benjamin Grzesik, Mohamed Khalil Ben Larbi, Enrico Stoll, Tom Haylock, Harald Konstanski, Kattia Flores Pozo, Mirue Choi, Christian Schurig, and Sándor P. Fekete. 'Automated Data Retrieval from Large-Scale Distributed Satellite Systems'. In: *15th IEEE International Conference on Automation Science and Engineering, CASE 2019, Vancouver, BC, Canada, August 22-26, 2019*. IEEE, 2019
5. Sándor P. Fekete and Dominik Krupke. 'Practical Methods for Computing Large Covering Tours and Cycle Covers with Turn Cost'. In: *Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments, ALENEX 2019, San Diego, CA, USA, January 7-8, 2019*. Ed. by Stephen G. Kobourov and Henning Meyerhenke. SIAM, 2019 (excluding the part on integer programming, which originated from the master's thesis and, thus, cannot be included in the dissertation.)
6. Sándor P. Fekete, Alexander Hill, Dominik Krupke, Tyler Mayer, Joseph S. B. Mitchell, Ojas Parekh, and Cynthia A. Phillips. 'Probing a Set of Trajectories to Maximize Captured Information'. In: *18th International Symposium on Experimental Algorithms, SEA 2020, June 16-18, 2020, Catania, Italy*. Ed. by Simone Faro and Domenico Cantone. Vol. 160. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020
7. Aaron T. Becker, Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, Christian Rieck, Christian Scheffer, and Arne Schmidt. 'Tilt Assembly: Algorithms for Micro-Factories that Build Objects with Uniform External Forces'. In: *28th International Symposium on Algorithms and Computation, ISAAC 2017, December 9-12, 2017, Phuket, Thailand*. Ed. by Yoshio Okamoto and Takeshi Tokuyama. Vol. 92. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017
8. Aaron T. Becker, Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, Christian Rieck, Christian Scheffer, and Arne Schmidt. 'Tilt Assembly: Algorithms for Micro-factories That Build Objects with Uniform External Forces'. In: *Algorithmica* 82.2 (2020)
9. Theorem 4 of Phillip Keldenich, Sheryl Manzoor, Li Huang, Dominik Krupke, Arne Schmidt, Sándor P. Fekete, and Aaron T. Becker. 'On Designing 2D Discrete Workspaces to Sort or Classify Polyominoes'. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018
10. Aaron T. Becker, Sándor P. Fekete, Li Huang, Phillip Keldenich, Linda Kleist, Dominik Krupke, Christian

- Rieck, and Arne Schmidt. 'Targeted Drug Delivery: Algorithmic Methods for Collecting a Swarm of Particles with Uniform, External Forces'. In: *2020 IEEE International Conference on Robotics and Automation, ICRA 2020, Paris, France, May 31 - August 31, 2020*. IEEE, 2020
11. Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Joseph S. B. Mitchell. 'Area-Optimal Simple Polygonalizations: The CG Challenge 2019'. In: *CoRR abs/2111.07304* (2021). arXiv: [2111.07304](https://arxiv.org/abs/2111.07304)
 12. Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Joseph S. B. Mitchell. 'Computing Convex Partitions for Point Sets in the Plane: The CG: SHOP Challenge 2020'. In: *CoRR abs/2004.04207* (2020). arXiv: [2004.04207](https://arxiv.org/abs/2004.04207)
 13. Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Joseph S. B. Mitchell. 'Computing Coordinated Motion Plans for Robot Swarms: The CG: SHOP Challenge 2021'. In: *CoRR abs/2103.15381* (2021). arXiv: [2103.15381](https://arxiv.org/abs/2103.15381)

The selective nature of the thesis content is further illustrated by the fact that there have been numerous other papers that were co-authored preceding or concurrent to this dissertation, but are not part of it.

- ▶ Judith Bernett, Dominik Krupke, Sepideh Sadegh, Jan Baumbach, Sándor P. Fekete, Tim Kacprowski, Markus List, and David B. Blumenthal. 'Robust disease module mining via enumeration of diverse prize-collecting Steiner trees'. In: *Bioinformatics* (Jan. 2022)
- ▶ Mohamed Ben Larbi, Kattia Pozo, Mirue Choi, Tom Haylok, Benjamin Grzesik, Andreas Haas, Dominik Krupke, Harald Konstanski, Volker Schaus, Sándor Fekete, Christian Schurig, and Enrico Stoll. 'Towards the Automated Operations of Large Distributed Satellite Systems. Part 2: Classifications and Tools'. In: *Advances in Space Research* 67 (Sept. 2020)
- ▶ Mohamed Ben Larbi, Kattia Pozo, Tom Haylok, Mirue Choi, Benjamin Grzesik, Andreas Haas, Dominik Krupke, Harald Konstanski, Volker Schaus, Sándor Fekete, Christian Schurig, and Enrico Stoll. 'Towards the Automated Operations of Large Distributed Satellite Systems. Part 1: Review and Paradigm Shifts'. In: *Advances in Space Research* 67 (Aug. 2020)
- ▶ Volker Schaus, Dominik Krupke, Mohamed Ben Larbi, Andreas Haas, Benjamin Grzesik, Jonas Radtke, Sándor Fekete, and Enrico Stoll. 'Automated Constellation Management With Self-Regulating Data-Economic Actors'. In: *70th International Astronautical Congress (IAC)*. Oct. 2019
- ▶ Sándor P. Fekete and Dominik Krupke. 'Covering Tours and Cycle Covers with Turn Costs: Hardness and Approximation'. In: *Algorithms and Complexity - 11th International Conference, CIAC 2019, Rome, Italy, May 27-29, 2019, Proceedings*. Ed. by Pinar Heggernes. Vol. 11485. Lecture Notes in Computer Science. Springer, 2019
- ▶ An Nguyen, Dominik Krupke, Mary Burbage, Shriya Bhatnagar, Sándor P. Fekete, and Aaron T. Becker. 'Using a UAV for Destructive Surveys of Mosquito Population'. In: *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*. IEEE, 2018
- ▶ Aaron T. Becker, Mustapha Debboun, Sándor P. Fekete, Dominik Krupke, and An Nguyen. 'Zapping Zika with a Mosquito-Managing Drone: Computing Optimal Flight Patterns with Minimum Turn Cost (Multimedia Contribution)'. In: *33rd International Symposium on Computational Geometry, SoCG 2017, July 4-7, 2017, Brisbane, Australia*. Ed. by Boris Aronov and Matthew J. Katz. Vol. 77. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017
- ▶ Sándor P. Fekete, Andreas Haas, Michael Hemmer, Michael Hoffmann, Irina Kostitsyna, Dominik Krupke, Florian Maurer, Joseph S. B. Mitchell, Arne Schmidt, Christiane Schmidt, and Julian Troegel. 'Computing nonsimple polygons of minimum perimeter'. In: *J. Comput. Geom.* 8.1 (2017)
- ▶ Arun V. Mahadev, Dominik Krupke, Jan-Marc Reinhardt, Sándor P. Fekete, and Aaron T. Becker. 'Collecting a swarm in a grid environment using shared, global inputs'. In: *IEEE International Conference on Automation Science and Engineering, CASE 2016, Fort Worth, TX, USA, August 21-25, 2016*. IEEE, 2016
- ▶ Sándor P. Fekete, Andreas Haas, Michael Hemmer, Michael Hoffmann, Irina Kostitsyna, Dominik Krupke, Florian Maurer, Joseph S. B. Mitchell, Arne Schmidt, Christiane Schmidt, and Julian Troegel. 'Computing Nonsimple Polygons of Minimum Perimeter'. In: *Experimental Algorithms - 15th International Symposium, SEA 2016, St. Petersburg, Russia, June 5-8, 2016, Proceedings*. Ed. by Andrew V. Goldberg and Alexander S. Kulikov. Vol. 9685. Lecture Notes in Computer Science. Springer, 2016
- ▶ Dominik Krupke, Maximilian Ernestus, Michael Hemmer, and Sándor P. Fekete. 'Distributed cohesive

control for robot swarms: Maintaining good connectivity in the presence of exterior forces'. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015, Hamburg, Germany, September 28 - October 2, 2015*. IEEE, 2015

- ▶ Dominik Krupke, Michael Hemmer, James McLurkin, Yu Zhou, and Sándor P. Fekete. 'A parallel distributed strategy for arraying a scattered robot swarm'. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015, Hamburg, Germany, September 28 - October 2, 2015*. IEEE, 2015
- ▶ Björn Bankowski, Thimo Clausen, Dirk Ehmen, Maximilian Ernestus, Henning Hasemann, Tobias Jura, Alexander Kröller, Dominik Krupke, and Marco Nikander. 'Panic Room: Experiencing Overload and Having Fun in the Process'. In: *Distributed, Ambient, and Pervasive Interactions - Second International Conference, DAPI 2014, Held as Part of HCI International 2014, Heraklion, Crete, Greece, June 22-27, 2014. Proceedings*. Ed. by Norbert A. Streitz and Panos Markopoulos. Vol. 8530. Lecture Notes in Computer Science. Springer, 2014

Contents

Contents	xiii
1. Introduction	1
I. OPTIMIZATION PROBLEMS OF SATELLITE SWARMS	5
2. Minimum Scan Cover	9
2.1. Introduction	9
2.1.1. Preliminaries	10
2.1.2. Related Work	12
2.1.3. Overview	13
2.2. Complexity in 1D	14
2.2.1. Bounds	15
2.2.2. Approximation Hardness	18
2.2.3. Polynomial Cases for MSC-MS	19
2.3. Hardness in 2D	19
2.3.1. MSC-MS	20
2.3.2. MSC-TE and MSC-BE	23
2.4. Approximations in 2D	29
2.4.1. Bipartite Graphs in 2D	29
2.4.2. Graphs with Bounded Chromatic Number	32
2.5. 3D and Abstract	34
2.6. Computational Study	37
2.6.1. Optimal Solutions	37
2.6.2. Good Solutions	40
2.7. Conclusion	44
3. Angular Freeze-Tag	47
3.1. Introduction	47
3.1.1. Related Work	48
3.1.2. Overview	49
3.2. Hardness	49
3.3. Approximation Algorithm	52
3.4. Exact Algorithms	52
3.4.1. Mixed Integer Programming	53
3.4.2. Constraint Programming	55
3.4.3. How Does CP-SAT Work?	65
3.5. Heuristics	68
3.5.1. Random	68
3.5.2. Greedy	69
3.6. Evaluation	72
3.6.1. Benchmark	73
3.6.2. Optimal Solutions	73
3.6.3. Good Solutions	74
3.7. Adaptions for BAFT	75
3.7.1. Adapting the CP	75

3.7.2. Greedy Algorithm	76
3.8. Conclusion	77
4. Automated Data Retrieval from Large-Scale Distributed Satellite Systems	79
4.1. Introduction	79
4.2. Related Work	81
4.2.1. Decentralized Auction-Based Scheduling	82
4.3. Preliminaries	82
4.4. Concept	83
4.5. Auctioneer’s Algorithm	84
4.5.1. Optimal Bid Selection	85
4.5.2. Price Estimation	85
4.6. Bidding Strategy	87
4.7. Experimental Evaluation	87
4.7.1. Greedy Algorithm	88
4.7.2. Experimental Results	88
4.7.3. Contact Pauses	89
4.8. Conclusion	89
II. PARTIAL COVERAGE PATH PLANNING	91
5. Engineering an Approximation Algorithm	95
5.1. Introduction	95
5.1.1. Related Work	96
5.1.2. Overview	98
5.2. Preliminaries	99
5.3. Efficient Implementation	99
5.3.1. Atomic Strip Covers	100
5.3.2. Matching	102
5.3.3. Partial Coverage	102
5.3.4. Tours	103
5.3.5. Other Grids	103
5.4. Evaluation	104
5.5. Conclusion	107
6. Generalization to Polygonal Areas	109
6.1. Introduction	109
6.2. Problem Definition	111
6.3. Related Work	112
6.3.1. Full Coverage	113
6.3.2. Partial Coverage	113
6.3.3. Touring Costs	114
6.4. Generalized Algorithm	115
6.4.1. Discretization	115
6.4.2. Fractional Solution	117
6.4.3. Atomic Strips	118
6.4.4. Matching	121
6.4.5. Local Optimization	122
6.4.6. Connecting Cycles	124
6.4.7. Local Optimization	126
6.4.8. Default Algorithm	128

6.5.	Grids and Meshes	128
6.5.1.	Regular Grids	129
6.5.2.	Meshes	134
6.5.3.	Comparison	138
6.6.	Evaluation	142
6.6.1.	Integralization	143
6.6.2.	Local Optimization	143
6.6.3.	Optimality Gap	146
6.6.4.	Objectives	146
6.6.5.	Runtime	148
6.7.	Conclusion	150
III. CAPTURING TRAJECTORIES		151
7.	Probing a Set of Trajectories to Maximize Captured Information	153
7.1.	Introduction	153
7.1.1.	Overview	154
7.1.2.	Related Work	155
7.2.	Preliminaries	156
7.3.	Analytical Results	156
7.3.1.	Complexity	157
7.3.2.	Approximation	160
7.4.	Algorithm Engineering	161
7.4.1.	Integer Programming	161
7.4.2.	Heuristics	164
7.4.3.	Generating Benchmark Instances	165
7.4.4.	Experimental Evaluation	165
7.5.	Conclusion	170
IV. TILT PROBLEMS		173
8.	Tilt Assembly	177
8.1.	Introduction	177
8.1.1.	Overview	179
8.1.2.	Related Work	179
8.2.	Preliminaries	181
8.3.	Constructibility of Simple Polyominoes	181
8.3.1.	A Key Lemma	182
8.3.2.	An Efficient Algorithm	184
8.3.3.	Pipelined Assembly	185
8.3.4.	Error Detection	187
8.4.	Optimization Variants	190
8.5.	Three-dimensional Shapes	193
8.6.	Computational Study	196
8.6.1.	Solving as SAT-Formula	196
8.6.2.	Constraint Programming	198
8.6.3.	Evaluation	198
8.7.	Conclusion	201

9. Targeted Drug Delivery	203
9.1. Introduction	203
9.1.1. Overview	204
9.1.2. Related Work	204
9.2. Preliminaries	205
9.3. Algorithmic Approaches	206
9.3.1. Complexity	206
9.3.2. Merging Two Particles	208
9.3.3. Reducing the Number of Particles	211
9.3.4. General Upper Bounds	212
9.4. Reinforcement Learning	212
9.4.1. Reward	214
9.4.2. Implementation	216
9.5. Evaluation	217
9.5.1. Oblivious Merging	220
9.6. Conclusions	220
V. CG:SHOP CHALLENGES	223
10. On Hosting the CG:SHOP Challenges	227
10.1. Introduction	227
10.2. Server Architecture	228
10.2.1. Frameworks and Libraries	228
10.2.2. Components	231
10.3. Workload and Tasks	234
10.3.1. Problem Selection	234
10.3.2. Development	235
10.3.3. Instance Selection	236
10.3.4. Support	236
10.3.5. Maintenance	236
10.3.6. Analysis	237
10.4. Instance Selection	237
10.5. Lessons Learned	241
11. CG:SHOP Challenge 2019	243
11.1. Introduction	243
11.1.1. Challenge Problem	243
11.1.2. Related Work	244
11.1.3. Outcomes	245
11.2. Pick's Theorem and Integrality	246
11.3. Approximation	247
11.4. Contest and Outcomes	248
11.4.1. Instances	248
11.4.2. Evaluation	249
11.4.3. Results	249
11.5. Conclusions	254
12. CG:SHOP Challenge 2020	255
12.1. Introduction	255
12.2. Related Work	255
12.3. Instances	256

12.4. Evaluation	257
12.5. Categories	257
12.6. Server and Timeline	258
12.7. Outcomes	258
12.8. Conclusions	259
13. CG:SHOP Challenge 2021	261
13.1. Introduction	261
13.2. The Challenge	261
13.2.1. The Problem	261
13.2.2. Related Work	262
13.2.3. Instances	264
13.2.4. Evaluation	266
13.2.5. Categories	266
13.2.6. Server and Timeline	266
13.3. Outcomes	266
13.4. Conclusions	268
Bibliography	273

Introduction

1.

The field of Computational Geometry gives rise to many practically relevant problems that are provably hard to solve, such as network optimization problems or trajectory planning. The field of algorithm engineering, on the other hand, provides us with many techniques to solve hard problems, sometimes even to optimality. Throughout this thesis, we use various algorithms and techniques such as linear and mixed integer programming, constraint programming, SAT-solvers, deep reinforcement learning, auction-based algorithms, genetic algorithms, etc., to solve various geometric optimization problems. Techniques that are relatively uncommon or novel, such as CP-SAT or deep reinforcement learning for optimization problems, are accompanied by an introduction and explanation.

This thesis consists of five parts (with a total of twelve chapters), whereby each part considers a different set of problems. Each part, and most chapters, can be read independently. However, it is recommended to read this thesis in the order in which it is presented as the organization was intentional and follows logically.

Part I considers optimization problems and algorithmic challenges, which have arisen with the advent of large satellite constellations (Figure 1.1). In Chapter 2 (Minimum Scan Cover) on page 9, we optimize a problem in which satellites have to establish a set of inter-satellite links with directed antennas that require costly rotations. Even when we simplify the satellites to static points in the Euclidean space, the problem shows interesting theoretical properties, including a strong relationship to the chromatic number. After a number of complexity and approximation results, we use mixed integer programming and constraint programming to compute optimal solutions. The SAT-based constraint programming solver CP-SAT shows a surprising strength in solving this problem. Additionally, we apply a set of (meta-)heuristics, including genetic algorithms, to compute good solutions and compare them with the performance of the approximation algorithms. In Chapter 3 (Angular Freeze-Tag) on page 47, we consider a similar problem in which we have to broadcast a message within a satellite swarm with directed antennas. After proving NP-hardness even for bipartite graphs in the plane and providing an approximation algorithm for one problem variant, we use similar techniques as in Chapter 2 to compute exact and good solutions. Again, CP-SAT shows a superior performance as compared to classical mixed integer programming, and we take a deeper look into its underlying techniques. In Chapter 4 (Automated Data Retrieval from Large-Scale Distributed Satellite Systems) on page 79, we optimize the usage of classical ground stations for downlinking data, and we try to minimize the data loss if there is not enough capacity for all satellites. We propose an auction-based algorithm that can also be extended to provide a ground-station-as-a-service-system. The resulting schedules of this approach prove to be superior to classical greedy strategies, evaluated in realistic simulations.

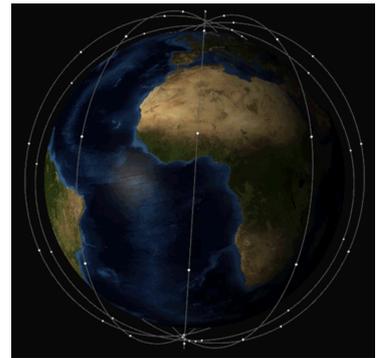


Figure 1.1: A large network of satellites.

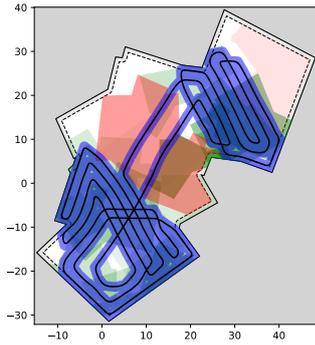


Figure 1.2.: A polygonal area with valuable (green) and difficult (red) regions. The trajectory is displayed in black, with the coverage in blue.

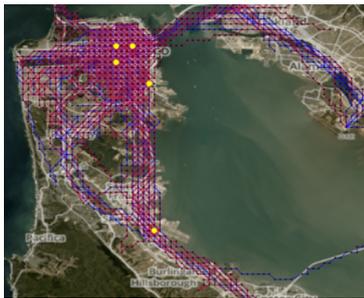


Figure 1.3.: Taxi trajectories and guards which capture as much of them as possible.

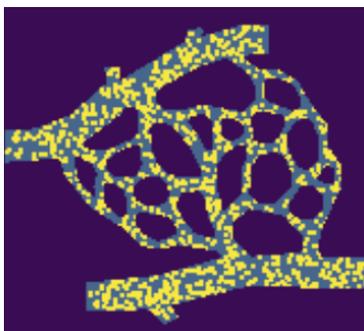


Figure 1.4.: The yellow particles are gathered by applying magnetic forces to them.

Part II deals with optimizing trajectories for (partially) covering an area, e.g., for aerial surveillance, a vacuum robot, or a harvester. We are given a polygonal area with valuable regions to cover as well as difficult regions that increase the energy consumption, see Figure 1.2 for an example. The task is to plan a trajectory for a circular tool that minimizes the touring costs (consisting of distances and turn angles) and the value of the missed area. In Chapter 5 (Engineering an Approximation Algorithm) on page 95, we begin with an abstract version of this problem in grid graphs; this essentially equals to the TRAVELING SALESMAN PROBLEM IN GRID GRAPHS WITH TURN COSTS. We already showed in previous work [18, 27], that the cycle cover relaxation is NP-hard, answering Problem 53 in the Open Problems Project [28], and provided an approximation algorithm. This approximation algorithm uses a linear programming relaxation to convert the problem into a minimum-weight perfect matching instance. In its original form, it can only solve relatively small instances in a reasonable time, but we engineer this algorithm to solve instances with over 300 000 vertices. The experimental evaluation also shows visibly better tours compared to the only other known approximation algorithm. In Chapter 6 (Generalization to Polygonal Areas) on page 109, we generalize this algorithm to optimize trajectories for polygonal instances, as described above. We show how to convert polygonal instances into discrete graphs in which we only have to compute a tour, and we optimize the tour by introducing a heuristic that computes optimal solutions for local parts with mixed integer programming.

Part III, which is comprised only of Chapter 7 (Probing a Set of Trajectories to Maximize Captured Information) on page 153, continues with trajectories but captures them instead of planning them. The TRAJECTORY CAPTURING PROBLEM asks for a placement of a limited number of guards onto intersecting trajectories such that the parts captured between any two guards are maximized. Such problems have applications in data collection and compression, see Figure 1.3. On the theoretical side, we prove NP-hardness even for orthogonal, straight-line trajectories, provide approximation algorithms, and show an unlimited integrality gap of the linear programming relaxation. On the practical side, we compute optimal solutions with mixed integer programming and evaluate a set of (meta-)heuristics.

Part IV returns to planning trajectories but this time for tiny particles, which have to be actuated by external forces, e.g., magnetic fields. Because of the similarity to *Labyrinth* (marble game) or Ball-in-a-maze puzzles, such problems are often referred to as *tilt problems* in the literature. These models find application in, e.g., the field of medicine controlling micro robots within the human body. In Chapter 8 (Tilt Assembly) on page 177, we search for construction plans to build specific shapes by iteratively inserting sticky particles. The difficulty is that new particles can only be added from the outside in a straight motion, which makes it impossible to build certain shapes. We provide complexity results and algorithms for various classes of shapes for deciding constructibility or maximizing the constructible part. Additionally, we show how to use a SAT-solver to compute construction sequences or show infeasibility for arbitrary shapes with more than 500 particles. In Chapter 9 (Targeted Drug Delivery) on page 203, we do not construct something, rather try to gather all particles, e.g., for targeted drug delivery. Here, the challenge is that all

particles move uniformly by the external force, and we need to utilize the environment to manipulate a particle swarm, as in Figure 1.4. We prove that the problem is NP-hard even in two-dimensional environments, but also provide algorithms with performance guarantees. However, the best command sequences in our experiments are computed by an approach that utilizes a deep reinforcement learning algorithm. We show how easy it has become to use popular reinforcement learning libraries to optimize combinatorial problems.

Part V takes a different form and discusses the CG:SHOP challenges. The CG:SHOP challenges are a yearly computational competition of CG Week; participants are provided with an optimization problem in Computational Geometry and a set of instances. Over a span of a few months, teams try to compute the best solutions and submit them through a web-interface, see Figure 1.5. In Chapter 10 (On Hosting the CG:SHOP Challenges) on page 227, we provide first-hand experience of hosting these competitions, and the first three challenges are summarized in Chapter 11 (CG:SHOP Challenge 2019) on page 243, Chapter 12 (CG:SHOP Challenge 2020) on page 255, and Chapter 13 (CG:SHOP Challenge 2021) on page 261.

The screenshot shows the CG:SHOP 2022 webpage. At the top, there is a blue navigation bar with 'CG:SHOP' and 'Help' on the left, and 'Competitions', 'Knapik', and a user profile icon on the right. The main content area is titled 'CG:SHOP 2022' and includes details about the organizers (Philip Kiderohf and Dominik Knapik), the start date (Jan 19, 2022), and the end date (Jan 19, 2022). There are two news items: 'We have started to verify your uploads...' and 'CG:SHOP 2022 competition instances released'. Below the news items is a navigation menu with 'Announcement', 'Rules', 'Instance Format', 'Your Teams', and 'Team Invitations'. The main announcement is titled 'Minimum Partition into Plane Subgraphs' and describes the challenge as a difficult geometric optimization problem.

Figure 1.5.: Webpage of CG:SHOP.

Part I.

**OPTIMIZATION PROBLEMS OF
SATELLITE SWARMS**

This part of the thesis considers three optimization problems in the context of satellite swarms. It provides basic research on underlying geometric optimization challenges induced by inter-satellite links that require costly rotations, and also provides a practical algorithm for scheduling contact windows on sparse ground stations.

In Chapter 2, we consider the MINIMUM SCAN COVER problem, which is motivated by optimizing a set of direct inter-satellite communications involving costly rotations for the adjustment of the directed antennas. We show that the problem is intrinsically difficult even when abstracting the satellite to static points in Euclidean space and abstracting the antenna beams to rays. Minimizing the makespan reveals a close connection to the chromatic number in one-dimensional instances, preventing any constant-factor approximation for general instances. In this way, we can also improve the bound on the directed minimum cut cover number. Minimizing the energy consumption of the swarm or of single satellites also shows to be NP-hard. On the positive side, we provide approximation algorithms whose factors depend on the chromatic number in 2D and on the arboricity for other metrics. An empirical study on computing exact and heuristic solutions rounds off this chapter.

The ANGULAR FREEZE-TAG problem in Chapter 3 is of a similar nature as MINIMUM SCAN COVER, but instead of scheduling a set of concrete communications, we are interested in efficiently distributing a message. We start by looking at complexity results and then focus on computing exact and heuristic solutions. We pay special attention to constraint programming, and we investigate how well CP-SAT performs on such geometric problems. While mixed integer programming solvers like Gurobi or CPLEX are already well established in solving geometric problems, the more generic constraint programming solvers are currently scarcely used. The superior performance of CP-SAT on the ANGULAR FREEZE-TAG and MINIMUM SCAN COVER problems, however, shows that satisfiability-based solvers can be a serious alternative when linear programming-based solvers fail due to weak relaxations.

Chapter 4 closes the section on satellites with a more practical approach on scheduling contact windows on ground stations with a market-based strategy. While inter-satellite-based communication can improve the communication of satellites in the future, most satellites currently only support ground stations-based communication. If we reach a point at which the available ground stations no longer provide enough downlink capacity, we will need algorithms that can prioritize the most important data. We discuss a corresponding algorithmic framework that uses auctions to fairly distribute the available slots and evaluate its performance in a realistic simulation. This approach can also be adapted to sell resources in a *Ground Station as a Service* system.

Minimum Scan Cover*

2.

In this chapter, we study a natural graph optimization problem that arises from transition costs between incident edges, e.g., by inter-satellite communication with directed antennas. First, we investigate in depth the theoretical complexity and approximation possibilities for various objectives, especially for graphs embedded in the plane. For minimizing the makespan, we prove a strong relationship to the chromatic number of the underlying graph. Afterward, we consider the practical complexity of the problem. While mixed integer programming fails, a SAT-based constraint programming approach can solve medium-sized instances to optimality. Additionally, we analyze the solution quality of the approximation algorithms and various heuristics.

2.1. Introduction

During the recent years, the size of satellite constellations has been greatly increasing, with much larger constellations still planned. One of the critical bottlenecks is expected to be the communication between the satellites, which is currently primarily implemented via ground stations. This approach, however, is unlikely to scale such that inter-satellite links (ILS) should be considered. These can be implemented with directed, paraboloid antennas or laser beams, see Figure 2.1 for an example. In both cases, the satellites have to adjust, i.e., rotate, toward each other to establish the connection.

In many network optimization problems, where different locations need to be connected, the objective is to minimize the geometric distance between the involved vertices. For our scenario, the distance is of lesser importance as long as the line-of-sight is not disturbed. We are faced with a different class of optimization problem, where the objective function is not based on edge weights, but on the adjustment costs between two edges. Problems of this type do not only arise from long-distance communication; they also come into play when astro- and geophysical measurements are to be performed, in which groups of spacecraft can determine physical quantities not just at their current locations, but also along their common line of sight; see [29] for a description.

We consider the problem of how to schedule a set of direct bidirectional communications, such that the overall timetable is as efficient as possible. The problem, MINIMUM SCAN COVER WITH ANGULAR COSTS (MSC), asks to establish a collection of connections between a given set of locations, described by a graph $G = (V, E)$ that is embedded in space. For any connection (or scan) of an edge, the two involved vertices need to face each other; changing the heading of a vertex to cover a different connection

2.1 Introduction	9
2.1.1 Preliminaries	10
2.1.2 Related Work	12
2.1.3 Overview	13
2.2 Complexity in 1D	14
2.2.1 Bounds	15
2.2.2 Approximation Hardness	18
2.2.3 Polynomial Cases for MSC-MS	19
2.3 Hardness in 2D	19
2.3.1 MSC-MS	20
2.3.2 MSC-TE and MSC-BE	23
2.4 Approximations in 2D	29
2.4.1 Bipartite Graphs in 2D	29
2.4.2 Graphs with Bounded Chromatic Number	32
2.5 3D and Abstract	34
2.6 Computational Study	37
2.6.1 Optimal Solutions	37
2.6.2 Good Solutions	40
2.7 Conclusion	44

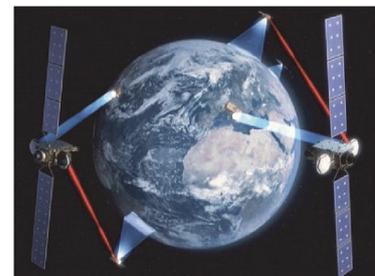


Figure 2.1.: Artist's rendition of the European Data Relay Satellite constellation architecture. Note the inter-satellite links shown in red. (Image credit: ESA)

* The content of this chapter was presented at SoCG 2020 [1] (full version has been published in SIDMA [2]) and at SEA 2021 [3]. Many thanks to Kevin Buchin, Sándor Fekete, Alexander Hill, Linda Kleist, Irina Kostitsyna, Roel Lambers, and Martijn Struijs for collaborations on these problems.

takes an amount of time proportional to the corresponding rotation angle. We assume this graph to be static for now, such that it can easily be described by a graph embedded in Euclidean space.

The first objective that we consider is MINIMUM MAKESPAN SCAN COVER (MSC-MS), in which we want to minimize the overall time (makespan) to establish all connections. The importance of conserving energy is captured in MINIMUM TOTAL ENERGY SCAN COVER (MSC-TE), where the goal is to minimize the sum of all rotation angles. In MINIMUM BOTTLENECK ENERGY SCAN COVER (MSC-BE), the task is to limit the energy used by any one vertex by minimizing the maximum total rotation at one vertex.

In this chapter, we give a comprehensive study of this optimization problem for these three objective functions, with a focus on two-dimensional geometric instances. Besides a theoretical analysis, providing complexity and approximation results, we also perform a computational study including additional heuristics and optimal solutions via mixed integer programming (MIP) and constraint programming (CP).

2.1.1. Preliminaries

For all considered versions of MINIMUM SCAN COVER (MSC), the *input* consists of a graph $G = (V, E)$. If not stated otherwise, G is a geometric (straight-line) embedded (not necessarily crossing-free) graph with $V \subset \mathbb{R}^d, d \in \{1, 2, 3\}$. We refer to the elements of V as *points* when their specific locations in \mathbb{R}^d are relevant; if we focus on graph properties, we may also refer to them as *vertices*. We denote the undirected edge between two vertices $u, v \in V$ by uv . For $v \in V$, we let $N(v) = \{u \in V : uv \in E\}$ be all vertices adjacent to v , and $E(v) = \{uv : u \in N(v)\}$ be all edges incident to v .

For two adjacent edges $uv, vw \in E(v)$, let $\alpha(uv, vw) \in [0, 180^\circ]$ denote the smaller angle between the lines supporting the segments uv and vw . The *output* for each problem is a *scan cover* $S : E \rightarrow \mathbb{R}^+$, such that for all pairs of adjacent edges e, e' , we have $|S(e) - S(e')| \geq \alpha(e, e')$. The geometric interpretation of a scan cover is that all points $v \in V$ have a *heading* that can change over time, and that if $S(uv) = t$ then u and v *face* each other at time t . In this case, we say that the edge uv is *scanned* at time t . Thus, the above condition on S guarantees that S complies with the necessary rotation time if rotation speed is bounded by 1.

We refer to the case in which we are given an abstract graph G , and α is an abstract metric cost function, as ABSTRACT MINIMUM SCAN COVER (Δ MSC). Note that this generalizes the Path-TSP for all objectives (see Observation 2.5.1), so the problem becomes intractable if the cost function α is not metric.

A *rotation scheme* describes the geometric change of headings of the vertices over a time interval of length T , i.e., it is a map $r : V \times [0, T] \mapsto [0^\circ, 360^\circ]$. The *total rotation angle* of a vertex v in r is the total amount that v rotates over $[0, T]$. For a given scan cover S , we are particularly interested in edges that are scanned consecutively. Therefore, let $v_S(e, e') = 1$ if e and e' share exactly one vertex v and the edge e' is scanned directly after e at v ; otherwise $v_S(e, e') = 0$.

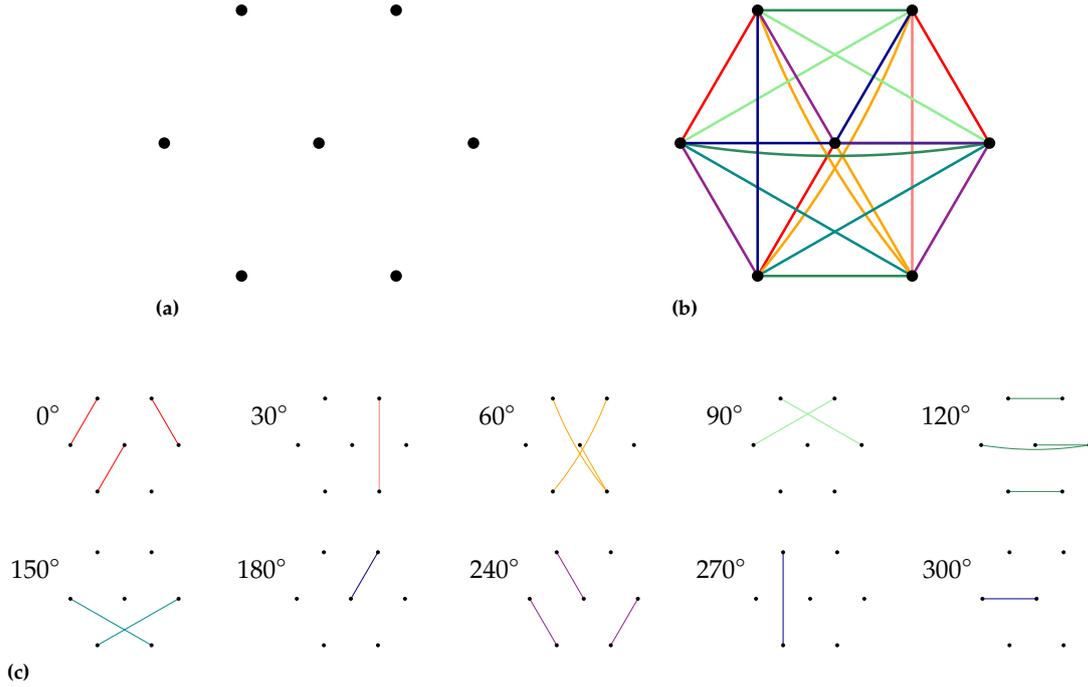


Figure 2.2.: (a) A set of seven points in \mathbb{R}^2 , for which all pairs of points shall communicate, i.e., (b) the complete graph K_7 needs to be scanned. (c) A sequence of edge scans. The scanned edges at 30° and 270° can also be performed in parallel at either time. However, the resulting scan cover has the same makespan. The nice 30° steps of the solutions are due to the regular structure of the instance. Other instances can look much more chaotic.

We consider the following three problems, defined by their respective objectives. For a given graph $G = (V, E)$ with vertices in the plane, find a scan cover S with

- Minimum Makespan (MSC-MS):

$$\min \max_{e \in E} S(e) \quad (2.1)$$

- Minimum Total Energy (MSC-TE):

$$\min \sum_{v \in V} \sum_{e, e' \in E(v)} \alpha(e, e') \cdot v_S(e, e') \quad (2.2)$$

- Minimum Bottleneck Energy (MSC-BE):

$$\min \max_{v \in V} \sum_{e, e' \in E(v)} \alpha(e, e') \cdot v_S(e, e') \quad (2.3)$$

Concentrating on the expensive and algorithmically challenging part of efficient rotations between the edges, we do not fix the initial heading of the satellites. In fact, all algorithms can be easily adapted to handle fixed initial headings. Furthermore, for every of the three objectives, an f -approximation can be converted into an $(f + 1)$ -approximation for the problem variant with fixed initial headings.

Figure 2.2 illustrates a minimum scan cover for a point set in the plane that can be scanned in 300° with eleven discrete time steps. We do not visualize the actual rotations as only the edge order is of importance. In fact, a scan cover is completely determined by an edge order: It is always optimal to directly rotate between two consecutive scans via the smallest

angle for all three objectives. For each edge sequence e_1, \dots, e_m , the best scan cover scanning the edges in this order can be computed by

$$S(e_1) = 0 \text{ and } S(e_i) = \max\{S(e_j) + \alpha(e_i, e_j) \mid j < i, e_i \cap e_j \neq \emptyset\} \text{ for } i > 1.$$

For a vertex v , we denote by $\Lambda(v)$ the minimum angle, such that a cone of this angle with apex v contains all edges in $E(v)$. We call such a cone a Λ -cone of v and call the complement of such a cone an *outer* cone of v . The Λ -cone will be a useful lower bound for the minimum necessary rotation of a vertex to scan all its edges. A Λ -cover is a scan cover for which every vertex v rotates in a single direction, either clockwise or counterclockwise, with a total rotation angle equal to $\Lambda(v)$. Note that different vertices can rotate in different directions. A Λ -cover minimizes both the MSC-TE and MSC-BE objectives.

Future Work 2.1.

If the communication between two satellites does not have to be bi-directional, one can also use multi-hop connections. Allowing a (directed) message also be forwarded by relay satellites can significantly reduce the necessary rotations. How does this change the characteristic of this problem?

2.1.2. Related Work

The use of directional antennas has introduced a number of geometric questions. Carmi et al. [30] study the α -MST problem, which arises from finding orientations of directional antennas with α -cones, such that the connectivity graph yields a spanning tree of minimum weight, based on bidirectional communication. They prove that for $\alpha < \pi/3$, a solution may not exist, while $\alpha \geq \pi/3$ always suffices. See Aschner and Katz [31] for more recent hardness proofs and constant-factor approximations for some specific values of α .

Many other geometric optimization problems deal with turn cost. Arkin et al. [32, 33] show hardness of finding an optimal milling tour with turn cost, even in relatively constrained settings, and give a 2.5-approximation algorithm for obtaining a cycle cover, yielding a 3.75-approximation algorithm for tours. The complexity of finding an optimal cycle cover in a two-dimensional grid graph was stated as *Problem 53* in *The Open Problems Project* [28] and shown to be NP-complete in [18], which also provides constant-factor approximations; practical methods and results are given in [5], and visualized in the video [20]. The second part of this thesis continues on this problem.

Finding a fastest roundtrip for a set of points in the plane for which the travel time depends only on the turn cost is called the ANGULAR METRIC TRAVELING SALESMAN PROBLEM. Aggarwal et al. [34] prove hardness and provide an $O(\log n)$ approximation algorithm for cycle covers and tours that works even for distance costs and higher dimensions. For the abstract version on graphs in which “turns” correspond to weighted changes between edges, Fellows et al. [35] show that the problem is fixed-parameter tractable in the number of turns, the treewidth, and the

maximum degree. Fekete and Woeginger [36] consider the problem of connecting a set of points by a tour in which the angles of successive edges are constrained.

Our problem also draws connections to other graph optimization problems. In particular, for each point in time, the set of scanned edges induces a bipartite graph. Therefore, one approach for scanning all edges of the given graph is to partition it into a small number of bipartite graphs, each corresponding to the set of edges separated by the *cut* induced by a partition of vertices into two non-trivial sets. This problem is also known as the MINIMUM CUT COVER PROBLEM: Find the minimum number of cuts to cover all edges of a graph. Loulou [37] shows that for complete graphs, an optimal solution consists of $\lceil \log_2 |V| \rceil$ cuts. Motwani and Naor [38] prove that, unless $P = NP$, the problem on general graphs is not approximable within 1.5 of the optimum, or $OPT + \varepsilon \log |V|$ for some $\varepsilon > 0$ in absolute terms, due to a direct relationship with GRAPH COLORING. Hoshino [39] considers practical methods based on integer programming and heuristics for cut covers. Chuzhoy and Khanna [40] show that the directed version of covering a directed graph by the minimum number of directed cuts is also an NP-hard problem.

On the application side, Korth et al. [29] describe the use of tomography (i.e., determining physical phenomena by measuring aggregated effects along a ray between two sensors) in the context of astrophysics. Using multiple sensors (e.g., satellites) for performing efficient measurements is one of the motivations for the algorithmic work in this chapter. Scheduling satellite communication has received a growing amount of attention, corresponding to the increasing size of satellite swarms.

In the context of scheduling, Allahverdi et al. [41–43] provide a nice and comprehensive survey on scheduling variants with sequence-dependent setup costs. Sotskov et al. [44] consider a scheduling variant that can directly be expressed as vertex coloring. In the context of earth observation, Li et al. [45] and Augenstein et al. [46] describe MIPs and heuristics to schedule image acquisition and downlink for satellites for which rotation and setup costs are taken into account.

2.1.3. Overview

In Section 2.2 (Complexity in 1D) on the following page, we show that MSC-TE and MSC-BE can be efficiently solved in 1D (Theorem 2.2.1) while MSC-MS corresponds to a minimum directed cut cover and has a strong correlation to the chromatic number. Additionally, we provide an improved upper bound of $\lceil \log_2 \chi(G) + 1/2 \cdot \log_2 \log_2 \chi(G) + 1 \rceil$ (Theorem 2.2.2 and Corollary 2.2.3) for the minimum directed cut cover number, which is essentially tight in general; even for directed acyclic graphs corresponding to minimum scan covers (Lemma 2.2.4) this is in the right order of magnitude. This implies that, unless $P = NP$, there exists no constant-factor approximation even in 1D (Theorem 2.2.6). Nevertheless, we show that instances in which the underlying graphs are bipartite or complete graphs can be solved in polynomial time (Observations 2.2.7 and 2.2.8).

While the objectives differ strongly in 1D, bipartite instances in 2D have a comparable complexity. In Section 2.3 (Hardness in 2D) on page 19, we

prove that bipartite instances in 2D are hard to approximate better than $3/2$ for MSC-MS (Theorem 2.3.2) and 1.04 for MSC-BE (Corollary 2.3.10) as well as NP-hardness for MSC-TE (Theorem 2.3.9). In Section 2.4 (Approximations in 2D) on page 29, we provide a 4.5-approximation (Theorem 2.4.3) for MSC-MS and a 2-approximation for the other two objectives (Theorem 2.4.4). More generally, we present an $O(C)$ -approximation for a k -colored graph with $k \leq \chi(G)^C$ (Theorem 2.4.8) for MSC-MS. This has immediate consequences for several interesting graph classes, e.g., the scan time of graphs in 1D and 2D lies in $\Theta(\log_2 \chi(G))$, and there exist constant factor approximations for various types of instances (Corollaries 2.4.9 and 2.4.10).

In Section 2.5 (3D and Abstract) on page 34, we consider all objectives in 3D and the abstract version Δ MSC. In contrast to 2D, the minimum makespan in 3D may exceed $O(\log_2 n)$ (Observation 2.5.4). Complementary to the fact that Δ MSC for stars is equivalent to path-TSP and thus NP-hard, we provide a 2.5-approximation of Δ MSC for trees (Theorem 2.5.5) on all three objectives. This yields an $O(\tau)$ -approximation for graphs with arboricity τ (Theorem 2.5.6).

Our practical study in Section 2.6 (Computational Study) on page 37 considers optimal solutions as well as heuristic solutions. For optimal solutions, we develop three mixed integer programs (MIPs), as well as two constraint programs (CPs) and evaluate their practical performance on a suite of benchmark instances. Solving instances of MSC-TE and MSC-BE to provable optimality turns out to be quite difficult; for MSC-MS, we are able to solve instances with up to 300 edges, based on one CP. This is surprising as MSC-MS is much harder on the theoretical side due to the connection to GRAPH COLORING that does not exist for MSC-TE and MSC-BE. In addition, we compare the solution quality of four (meta-)heuristics and the approximation algorithms on larger instances with up to 800 edges. In our experiments, a genetic algorithm and the intermediate solution after timeout of one CP produce the best solutions.

2.2. Complexity of One-Dimensional Point Sets

In the one-dimensional case, all vertices lie on a single line L . Therefore, an instance can be described by a graph $G = (V, E)$ and a total order of the vertices $<_L$ on L . We assume this line to be horizontal, so vertices face either left or right when scanning an edge. Moreover, scan times can be restricted to discrete multiples of 180° . This allows us to encode the headings of a vertex v at these time steps by a 0-1-vector $s(v)$, where a right heading is denoted by 0, and a left one by 1; we denote by $s_i(v)$ the i th bit of $s(v)$. Then a scan cover with N steps of $(G, <_L)$ is an assignment $s: V \rightarrow \{0, 1\}^N$, such that for every edge $uv \in E$, $u <_L v$, there exists an index $i \in [N]$ with $s_i(u) = 0$ and $s_i(v) = 1$. The makespan of such a scan cover is clearly $180^\circ(N - 1)$. For an example, consider Figure 2.3.

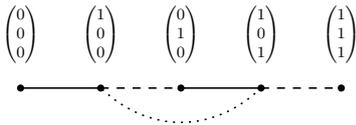


Figure 2.3. This instance can be scanned in three steps. However, two steps are not sufficient because the edges of each monotone path would need to be scanned in alternating time steps; making it impossible to scan the dotted edge if solid and dashed edges are already scanned alternately.

For MSC-TE and MSC-BE, we can actually solve these instances in polynomial time.

Theorem 2.2.1 *MSC-TE and MSC-BE in 1D are in P. Moreover, denoting by k the number of vertices with neighbors to both sides, the objective value is 0° for $k = 0$, while for $k > 0$ it is $180^\circ \cdot k$ for MSC-TE and 180° for MSC-BE.*

Proof. We assume that the vertices are placed on a horizontal line. We partition the vertices into two groups: those with a neighbor to only one side and those with neighbors to both sides. If the second group is empty, there is a trivial zero-cost solution for both objectives.

Thus, consider $k > 0$. Each of these vertices needs to rotate at least 180° , so the values $180^\circ \cdot k$ and 180° are lower bounds for MSC-TE and MSC-BE, respectively. The following strategy matches this lower bound: The vertices in the first group are headed to their neighbor and do not rotate. In the following we restrict our attention to the vertices in the second group. In the beginning, all of them are headed left. Then, from left to right, one after the other rotates such that it is headed right; the next vertex starts only after the completion of its predecessor. Note that whenever a vertex rotates, all edges to its left are scanned. Consequently, this yields a valid scan cover. \square

MSC-MS shows to be significantly harder due to a relation to the chromatic number, which MSC-TE and MSC-BE do not have.

2.2.1. Bounds Based on Chromatic Number and Cut Cover Number

In the following, we establish a strong relationship between the makespan of a MSC-MS in 1D and the chromatic number $\chi(G)$, which is closely linked to the cut cover number $c(G)$ of the involved graph $G = (V, E)$, i.e., the size of a smallest partition of the edge set into bipartite graphs. Motwani and Naor [38] show that

$$c(G) = \lceil \log_2 \chi(G) \rceil.$$

Because the scanned edges in each time step form a bipartite graph, a scan cover induces a cut cover. However, the resulting bipartite graphs have the additional property that for each vertex all neighbors are either smaller or larger with respect to $<_L$. Thus, not every cut cover corresponds to a scan cover. However, scan covers correspond to *directed* cut covers of the directed graph, induced by orienting the edges from left to right. Note that $\chi(G)$ is not influenced by directing the edges. Watanabe et al. [47] bound the directed cut cover number $\vec{c}(G)$ of a directed graph G :

$$\vec{c}(G) \leq \lceil \log_2 \chi(G) \rceil + \lceil \log_2 \lceil \log_2 \chi(G) + 1 \rceil \rceil$$

We improve this bound by showing an upper bound for the size of a smallest scan cover in terms of the chromatic number (and the cut cover number); this bound is best possible for the directed cut cover number as we explain later.

Theorem 2.2.2 For every graph G with $\chi(G) \geq 2$ and every ordering $<_L$ of the vertices, there exists a scan cover of $(G, <_L)$ with N steps such that

$$N \leq \lceil \log_2 \chi(G) + 1/2 \cdot \log_2 \log_2 \chi(G) + 1 \rceil \quad (2.4)$$

Proof. Consider a coloring of G with $C := \chi(G)$ colors and choose an N large enough such that $C \leq \binom{N}{\lfloor N/2 \rfloor}$. For $k := \lfloor N/2 \rfloor$, we consider the set of vectors $\{0, 1\}_k^N$ of length N with exactly k many 1's. We define a scan cover $s : V \rightarrow \{0, 1\}_k^N$, such that for all vertices of the same color, we assign the same vector, while vertices of different color obtain different vectors. Such an assignment exists, because the number of vectors, i.e., $\binom{N}{\lfloor N/2 \rfloor}$, is at least as large as the number of colors.

To see that s is a scan cover, consider a fixed but arbitrary edge uv of G . Because the vectors $s(u)$ and $s(v)$ differ but have the same number of 1's, they are *incomparable*, i.e., there exist i and j such that $s_i(u) = 0, s_i(v) = 1$ and $s_j(u) = 1, s_j(v) = 0$. Therefore, depending on the ordering of u and v on L , the edge uv is either scanned in step i or j .

It remains to show that defining $N := \lceil \log_2 C + 1/2 \cdot \log_2 \log_2 C + 1 \rceil$ satisfies $C \leq \binom{N}{\lfloor N/2 \rfloor}$. By a variant of Stirling's formula [48], it holds that

$$e^{1/(12n+1)} \leq \frac{n!}{\sqrt{2\pi n}(n/e)^n} \leq e^{1/(12n)}. \quad (2.5)$$

This implies that $\binom{N}{\lfloor N/2 \rfloor} \geq \sqrt{2/\pi N} \cdot 2^N \cdot e^{-\frac{1}{4N-1}}$, so it suffices to guarantee

$$\begin{aligned} C &\leq \sqrt{2/\pi N} \cdot 2^N \cdot e^{-\frac{1}{4N-1}} \\ \iff \log_2 C &\leq N + 1/2(1 - \log_2 \pi - \log_2 N) - \log_2 e/(4N-1). \end{aligned}$$

If $C \geq 3$, this holds for $N = \lceil \log_2 C + 1/2 \cdot \log_2 \log_2 C + 1 \rceil \geq 3$; in case of $C = 2$, it holds that $N = \lceil \log_2 C + 1/2 \cdot \log_2 \log_2 C + 1 \rceil = 2$, and thus $C \leq \binom{N}{\lfloor N/2 \rfloor}$. \square

Note that the assigned vectors in the proof of Theorem 2.2.2 are pairwise incomparable. Therefore, such an assignment yields a directed cut cover for all edge directions and thus a general bound on the directed cut cover number.

Corollary 2.2.3 For every directed graph G , the directed cut cover number is bounded by

$$\vec{c}(G) \leq \lceil \log_2 \chi(G) + 1/2 \cdot \log_2 \log_2 \chi(G) + 1 \rceil.$$

In fact, the bound in Corollary 2.2.3 is best possible for general directed graphs, because a cut cover of the complete bidirected graph corresponds to an assignment of pairwise incomparable vectors (and Sperner's theorem asserts that the used set of vectors is maximal). Figure 2.3 illustrates an example of a graph G and an ordering $<_L$ showing that the bound of Theorem 2.2.2 and Corollary 2.2.3 is also tight for some (directed acyclic) graphs with $\chi(G) = 3$. In the following, we show a general lower bound for our more special setting.

Lemma 2.2.4 For every constant C , there exists a graph G and an ordering $<_L$ such that $\chi(G) > C$ and the number N of steps in every scan cover of $(G, <_L)$ is at least

$$N \geq \lceil \log_2 \chi(G) + 1/4 \cdot \log_2 \log_2 \chi(G) \rceil. \quad (2.6)$$

Proof. Let $\ell \geq 4$ be an integer divisible by 4 and $n := 2^\ell$ such that $2^n > C$. We consider the Turan graph G on $n2^n$ vertices partitioned into 2^n independent sets of size n , see Figure 2.4a. Because G is a complete 2^n -partite graph, it holds that $\chi(G) = 2^n$. We place the vertices on the line, such that for a fixed $\{1, \dots, 2^n\}$ -coloring of G , there exist n disjoint intervals in which the colors appear in the order $1, \dots, 2^n$, as illustrated in Figure 2.4b.

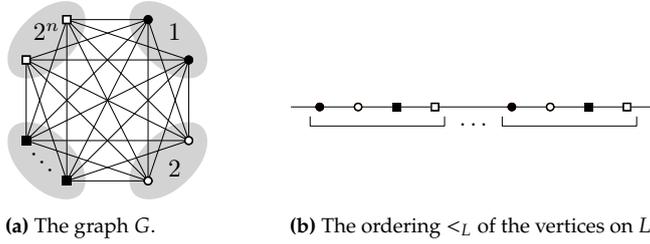


Figure 2.4: Illustration for the proof of Lemma 2.2.4.

For a contradiction, suppose that there exists a scan cover $s : V \rightarrow \{0, 1\}^k$ of $(G, <_L)$ with $k := \lceil \log_2 \chi(G) + 1/4 \cdot \log_2 \log_2 \chi(G) \rceil - 1 = n + \ell/4 - 1$ steps. Thus, the number of different vectors is $2^k = 2^{n-1}n^{1/4}$.

We say a color class has a *common vector* if some vector is assigned to at least $n^{3/4}$ of the n vertices of this color class. Let t denote the number of different color classes with a common vector; clearly, the number of color classes without a common vector is $2^n - t$. We show that $t \geq 1/2 \cdot 2^n$. Because vertices with the same vector will never face each other, each vector may only appear in one color class, i.e., the color classes induce a partition of the set of vectors. Consider the $2^n - t$ color classes (and their assigned vectors) in which every vector appears less than $n^{3/4}$ times. Let δ denote the average usage of vectors in these classes. Note that δ is lower bounded by the ratio of the number of vertices, namely $(2^n - t)n$, and the maximum number of remaining vectors, namely $2^k - t$. Consequently, $\delta \geq \frac{n2^n - tn}{2^k - t}$. Moreover, by our assumption, we consider color classes without common vectors and hence $\delta < n^{3/4}$. Therefore, we obtain the following chain of implications:

$$\begin{aligned} \delta < n^{3/4} &\implies \frac{n2^n - tn}{2^k - t} < n^{3/4} \\ &\iff t > 2^n \cdot \frac{1}{2(1 - n^{-1/4})} \\ &\implies t > 1/2 \cdot 2^n \end{aligned}$$

For each of these t color classes with a common vector, we choose a vector with a maximal number of appearances and introduce an interval on L from the first to the last occurrence. By the ordering of the vertices, every two vertices of the same color have a distance of at least 2^n , and hence

the interval spans at least $d = 2^n n^{3/4}$ vertices. On average, every vertex is contained in the following number of intervals

$$\frac{t \cdot d}{|V|} \geq \frac{1/2 \cdot 2^n \cdot 2^n n^{3/4}}{n 2^n} = \frac{2^n}{2n^{1/4}} = 2^{n-1} n^{-1/4}.$$

By the pigeonhole principle, there exists a set S of at least $2^{n-1} n^{-1/4}$ vectors with mutually intersecting intervals. We claim that any two vectors a and b of S are pairwise *incomparable*, i.e., there exist two indices i, j such that $a_i = 0, b_i = 1$ and $a_j = 1, b_j = 0$: Because the intervals intersect, among the four occurrences of a and b on $\langle L$, there exist three such that they appear alternating. To scan the corresponding edges, the vectors must be incomparable. Thus, there must exist $2^{n-1} n^{-1/4}$ pairwise incomparable vectors.

However, by Sperner's theorem, every set of vectors of length k contains at most $\binom{k}{\lfloor k/2 \rfloor}$ pairwise incomparable vectors and

$$\binom{k}{\lfloor k/2 \rfloor} \leq \sqrt{2/k\pi} \cdot 2^k \cdot (1 + 1/11) \leq 1/\sqrt{k} \cdot 2^k.$$

It remains to show that the number of necessary incomparable vectors exceeds this:

$$\frac{2^k}{\sqrt{k}} < \frac{2^{n-1}}{n^{1/4}} \iff n < k$$

This holds for $\ell > 4$ and yields a contradiction. For $\ell = 4$ it holds that $k = n$. Thus, each color class has a unique vector, all of which need to be incomparable, a contradiction. \square

2.2.2. No Constant-Factor Approximation in 1D

Theorem 2.2.2 implies the following.

Lemma 2.2.5 *For any constant C , a C -approximation for MSC-MS implies a polynomial-time algorithm for computing a coloring of graph G with $4^C \cdot k^C \cdot \sqrt{\log_2(\chi(G))}^C$ colors.*

Proof. Let ℓ^* denote the length of a minimum scan cover of G and $k := \chi(G)$. Then a C -approximation algorithm computes a scan cover of length $\ell \leq C \cdot \ell^*$. Theorem 2.2.2 implies that $C \cdot \ell^* \leq C \cdot \lceil \log_2 k + 1/2 \log_2 \log_2 k + 1 \rceil$, yielding a coloring with 2^ℓ colors. Thus, we obtain the following chain of inequalities:

$$\begin{aligned} 2^\ell &\leq 2^{C(\lceil \log_2 k + 1/2 \log_2 \log_2 k + 1 \rceil)} \leq 2^{C \cdot \log_2 k} \cdot 2^{1/2 \cdot C \cdot \log_2 \log_2 k} \cdot 2^{2C} \\ &\leq 4^C \cdot k^C \cdot \sqrt{\log_2(k)}^C. \end{aligned}$$

\square

Theorem 2.2.6 *Even in 1D, a C -approximation for MSC-MS for any constant $C \geq 1$ implies $P = NP$.*

Proof. Suppose MSC-MS allows a C -approximation for some constant $C \geq 1$. By Lemma 2.2.5, a C -approximation of MSC-MS in 1D implies that there is a polynomial-time algorithm for finding for every k -colorable graph G a coloring with $4^C \cdot k^C \cdot \sqrt{\log_2(k)}^C$ colors. Khot [49] showed that, for sufficiently large k , it is NP-hard to color a k -colorable graph with at most $k^{\log_2(k)/25}$ colors. However, for every C we can find a k such that $4^C \cdot k^C \cdot \sqrt{\log_2(k)}^C < k^{\log_2(k)/25}$. This yields a polynomial-time algorithm for an NP-hard problem, implying that $P = NP$. \square

2.2.3. Polynomial Cases for MSC-MS

Even though there is no constant-factor approximation in general for MSC-MS, we would like to note that bipartite and complete graphs in 1D can be solved in polynomial time.

Observation 2.2.7 *For instances of MSC-MS in 1D for which the underlying graph G is bipartite, there exists a polynomial-time algorithm for computing an optimal scan cover.*

Proof. We assume that $\chi(G) = 2$, otherwise there is no edge to scan. If for every vertex, all its neighbors lie either before or after it, G can be scanned within one step, which is clearly optimal. Otherwise, every scan cover needs at least two steps. By Theorem 2.2.2, there exists a scan cover with 2 steps. Because bipartite graphs can be colored in polynomial time, the proof of Theorem 2.2.2 provides a scan cover. \square

Observation 2.2.8 *For instances of MSC-MS in 1D for which the underlying graph G is a complete graph, there exists a polynomial-time algorithm for computing an optimal scan cover.*

Proof. Because every scan cover induces a cut cover and $c(G) = \lceil \log_2 n \rceil$, it suffices to provide a scan cover with this number of steps. To this end, we recursively scan the bipartite graphs induced by two vertex sets when split into halves with respect to the order $<_L$. \square

2.3. Hardness in 2D: Hardness of Bipartite Point Sets

By Theorem 2.2.6, we cannot hope for a constant-factor approximation for MSC-MS for general graphs. However, bipartite graphs in 1D can be solved in polynomial time. MSC-TE and MSC-BE can be solved efficiently at least in 1D, as seen in Theorem 2.2.1.

We now show that the added geometry of 2D makes the MSC-MS hard to approximate better than a factor of $3/2$, even for bipartite graphs. Afterwards, we show that also MSC-TE and MSC-BE become difficult, proving the NP-hardness of MSC-TE and that it is NP-hard to approximate bipartite graphs better than a factor of 1.04 for MSC-BE. Corresponding constant-factor approximation algorithms follow later in Section 2.4.

2.3.1. MSC-MS: Approximation Hardness for MSC-MS

As a stepping stone for the geometric case, we establish the following.

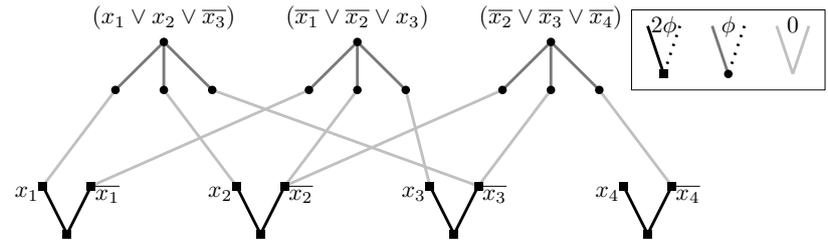
Lemma 2.3.1 *It is NP-hard to approximate Δ MSC-MS better than $3/2$, even for bipartite graphs.*

Proof. The proof is based on a reduction from NOT-ALL-EQUAL-3-SAT, for which a satisfying assignment fulfills the property that each clause has a true and a false literal, i.e., all false or all true is prohibited. The nice feature of this variant is that the negation of a satisfying assignment is also a satisfying assignment.

For every instance I of NOT-ALL-EQUAL-3-SAT, we construct a graph G_I and a cost function α , for which each edge pair has a transition cost of 0, ϕ , or 2ϕ . Thus, every optimal scan cover has discrete time steps at distance ϕ . We show that there exists a scan cover of (G_I, α) with three time steps, i.e., a scan time of 2ϕ , only if I is a satisfiable instance. Otherwise, every scan cover has at least four steps, i.e., a value of 3ϕ .

We now describe our construction of G_I , which is a special variant of a clause-variable-incidence graph. For an illustration, see Figure 2.5.

Figure 2.5.: Illustration of the graph G_I for the instance $I = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)$.



There are four types of vertices and three types of edges: For every clause C_i of I , we introduce a *clause gadget* consisting of a *clause vertex* and three *entry vertices*, each of which represents one of the literals appearing in the clause. The clause vertex is adjacent to every entry vertex of its gadget by a *clause edge*. For every variable x_i of I , we introduce a *variable vertex* and two *literal vertices*. The variable vertex is adjacent to both literal vertices via a *variable edge*. Moreover, for every entry vertex, we introduce an *incidence edge* to the literal vertex that it represents.

We define α as follows: The transition cost for any edge pair is ϕ if it contains a clause edge, 2ϕ if it contains a variable edge, and 0 otherwise. Note that every variable and clause edge are pairwise disjoint; hence this is well-defined.

We now show that if I is a satisfiable instance of NOT-ALL-EQUAL-3-SAT, then there exists a scan cover with three time steps: If a literal is set to true, then the variable edge of this literal vertex is scanned in the first time step and all remaining edges of the literal vertex in the third step. Likewise, if a literal is false, then its variable edge is scanned in the third step, and all other incident edges in the first step.

For each clause we choose one positive and negative literal to be *responsible*, the third literal is *intermediate*. The clause edges are scanned in the first, second, or third step, depending on whether their entry vertex corresponds to a responsible positive literal, an intermediate literal, or a responsible negative literal, respectively. Note that the edge pairs with transition costs of 2ϕ , namely the edges incident to literal vertices, are scanned in the first or third step. Thus, the value of this scan cover is 2ϕ . For an example, consider Figure 2.6.

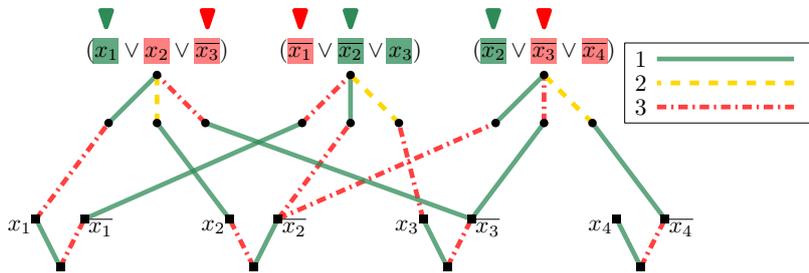


Figure 2.6.: Illustration of a scan cover of the graph G_I . Green edges are scanned in the first, yellow in the second, and red edges in the third step.

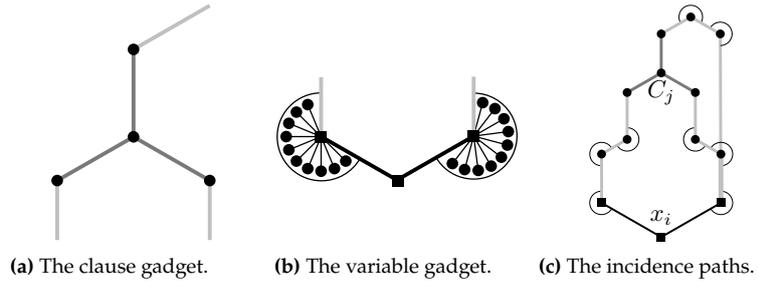
Now we consider the reverse direction and show that a scan cover with three time steps corresponds to a satisfying assignment of I . Because the transition cost of any two edges incident to a literal vertex is 2ϕ , each variable or incidence edge is scanned either in the first or third step. Therefore, we may define an assignment of I by setting the literals whose variable edge is scanned in the first time step to true. It remains to argue that in this assignment, every clause has a true and false literal. Note that the three edges of a clause gadgets, must be scanned at different time steps. Consequently, there exists a clause edge that is scanned in the first time step. Its adjacent incident edge is therefore scanned in the third step. This implies that the variable edge of the literal vertex is also scanned in the first time step and thus set to true. Likewise, the clause gadget in the third step corresponds to a false literal. Consequently, this assignment shows that I is a true-instance of NOT-ALL-EQUAL-3-SAT. \square

We now use Lemma 2.3.1 for showing hardness of bipartite graphs in the geometric version.

Theorem 2.3.2 *Even for bipartite graphs in 2D, a C -approximation for MSC-MS for any $C < 3/2$ implies $P = NP$.*

Proof. Suppose that there is a $(3/2 - \epsilon)$ -approximation for some $\epsilon > 0$. For every instance I of NOT-ALL-EQUAL-3-SAT, we can construct a graph G_I for MSC-MS in 2D such that it has a scan time of 240° if I is satisfiable, and a scan time of at least $360^\circ - \epsilon$ otherwise. We essentially use the same reduction as in the proof of Lemma 2.3.1. The idea is to embed the constructed graph G_I in the plane on a triangular grid such that the transition costs are reflected by the angle differences. Figure 2.7 depicts the gadgets.

Figure 2.7.: Illustration for the proof of Theorem 2.3.2 on how to embed the G_I into the plane with $\phi = 120^\circ$.



In particular, we choose $\phi = 120^\circ$. For each *clause gadget*, we create a star on four vertices with 120° angles between the edges. The incidence edges also leave with 120° from the three entry vertices. Consider Figure 2.7a for an illustration.

The vertices of the *variable gadget* can also easily be embedded with an angle difference of 120° . However, because the smaller angle between any two segments is at most 180° , we cannot directly construct angles of 240° . Therefore, we insert additional edges and vertices into the 240° angle with an angle difference of ε as illustrated in Figure 2.7b. If an incident vertex uses the smaller 120° angle, it would still need to cover the additional edges resulting in an overall turning angle of at least $360^\circ - \varepsilon = 3\phi - \varepsilon$.

To connect the *clause gadgets* with the *variable gadgets* we now need *incidence paths* instead of incidence edges. We use paths consisting of three edges where angle differences of 240° at the interior vertices are enforced as above, see Figure 2.7c. A path will propagate the decision by always scanning all odd or all even edges at the same time with a difference of 240° . Thus, the first and the last edge of the path are scanned at the same time.

Allowing the vertices to share their coordinates, we position all clause and variable gadgets at the same locations, respectively. This results in a constant number of coordinates. For unique coordinates, the gadgets are shifted horizontally, see Figure 2.8a. In particular, we consider an $O(1/(n+m))$ refinement of the grid, where n and m denote the number variables and clauses of I , respectively. In this grid, the gadgets can be shifted horizontally to neighboring grid positions; the vertices on the incidence paths are shifted to close positions. For an illustration

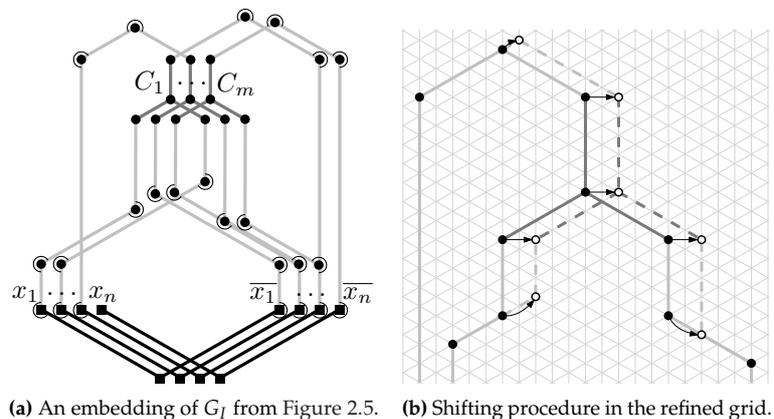


Figure 2.8.: Illustration for the construction with unique vertex coordinates.

consider Figure 2.8b.

A $(3/2 - \varepsilon)$ -approximation would now yield for a satisfiable instance a scan time of at most $(3/2 - \varepsilon) \cdot 240^\circ = 360^\circ - \varepsilon \cdot 240^\circ$ and decide the satisfiability because an unsatisfiable solution would have a scan time of at least $360^\circ - \varepsilon > 360^\circ - \varepsilon \cdot 240^\circ$. This is a contradiction to the NP-hardness of NOT-ALL-EQUAL-3-SAT. \square

2.3.2. MSC-BE and MSC-BE: Hardness

Next we show that for 2D instances of MSC-TE and MSC-BE, there does not exist an efficient algorithm, unless $P = NP$. Specifically, we show that MSC-TE and MSC-BE are NP-hard in 2D, even when the underlying graph $G = (V, E)$ is bipartite. Our proof is based on the observation that if a Λ -cover exists, any scan cover optimal for MSC-TE is a Λ -cover. If additionally all vertices have the same $\Lambda(v)$, any scan cover optimal for MSC-BE is a Λ -cover. We show finding a Λ -cover is NP-hard via a reduction from the NP-complete problem MONOTONE NOT-ALL-EQUAL 3-SATISFIABILITY (MNAE3SAT) [50, 51], defined as follows: Given a set of Boolean variables X and a set of clauses \mathcal{C} with at most 3 literals from X all of which are not negated, is there a 0/1-assignment to the variables in X , such that for each clause in \mathcal{C} , not all variables have the same value?

Given an instance I of the MNAE3SAT, we construct an MSC instance G_I (with the same $\Lambda(v)$ for all vertices) that has a Λ -cover if and only if I has a valid variable assignment. Recall that in a Λ -cover, the edges of each vertex are scanned in either clockwise or counter-clockwise order. We encode variable assignment by the rotation direction of the vertices in G_I in a Λ -cover. A variable is encoded by a subgraph that contains a set of vertices that have the same rotation direction in a Λ -cover, and a clause by a subgraph that contains three vertices that cannot all have the same rotation direction in a Λ -cover. We connect variables to clauses via wires, which are encoded by a subgraph that contains two vertices that have the same rotation direction in a Λ -cover. See Figure 2.9 for an example of the construction.

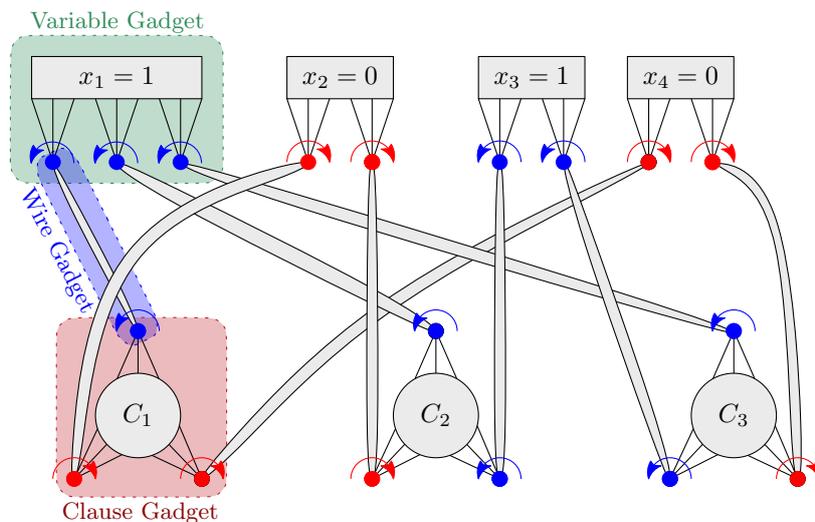


Figure 2.9: The constructed graph G_I for the instance $(x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_3 \vee x_4)$ of MNAE3SAT. The gadgets are drawn symbolically; also shown are the directions of the connector vertices corresponding to the satisfying assignment $x_1 = x_3 = 1, x_2 = x_4 = 0$.

The construction in the proof of Theorem 2.3.9 establishes a gap between optimal and suboptimal solutions, which implies a constant-factor

approximation lower bound for MSC-BE.

Constructing the gadgets We construct a *variable gadget* G_x (Figure 2.11) for each variable $x \in X$ and a *clause gadget* G_C (Figure 2.10b) for each clause $C \in \mathcal{C}$. For $x \in C$, we connect the gadgets G_x and G_C with a *wire gadget* G_w (Figure 2.12). The resulting graph G_I is symbolically shown in Figure 2.9. We construct both the variable gadget and wire gadget from smaller components called *wire fragments* G_f , see Figure 2.10a for an illustration.

We first state several observations that help with the construction of these gadgets.

Observation 2.3.3 *If an edge vw is the first or the last edge scanned in a Λ -cover, it bounds a minimal Λ -cone of vertex v and of vertex w .*

In the gadgets, we will prescribe the two edges bounding the Λ -cone of a vertex.

Observation 2.3.4 *Consider a straight-line drawing of a graph G . For every vertex v and every pair of consecutive edges e, e' at v , we can add edges incident to v (and new vertices of degree 1), such that in the resulting drawing, e and e' bound the Λ -cone of v .*

Observation 2.3.4 allows us to choose the maximal angle between consecutive edges. In the same manner, we can ensure that $\Lambda(v)$ is equal for all vertices v of the gadgets (excluding the newly added vertices of degree 1).

Next we construct the individual gadgets. Consider a wire fragment G_f as depicted in Figure 2.10a. Using Observation 2.3.4, we make sure that the Λ -cones correspond to the blue arcs. The vertices s, t, u can be shared between subgraphs with the following properties concerning their direction of rotation in a Λ -cover. Note that given a Λ -cover, we can obtain a second Λ -cover by reversing all directions.

Lemma 2.3.5 *The vertices s, t, u in the wire fragment G_f have the following properties.*

1. *In every Λ -cover of G_f , the vertices u and t rotate in the same direction, while s rotates in the opposite direction.*
2. *There exists a Λ -cover of G_f .*

Proof. Suppose we have a Λ -cover of G_f with scan order P . Note that one of the edges su, tv_2 is first in P , and the other last (these are the only edges bounding a minimal Λ -cone, see Observation 2.3.3). Suppose su is the first edge in P . Then s turns counterclockwise and u turns clockwise. Additionally, t turns clockwise, because tv_2 is the last edge. The other case, in which tv_2 is the first edge, is analogous.

The following scan order yields a Λ -cover of G_f in which s rotates counterclockwise, see also the edge labels in Figure 2.10a: $su, uv_3, v_2v_3, v_1v_2, uv_1, v_1v_5, sv_4, sv_5, tv_5, v_3v_5, v_3v_4, v_1v_4, tv_4, tv_2$. \square

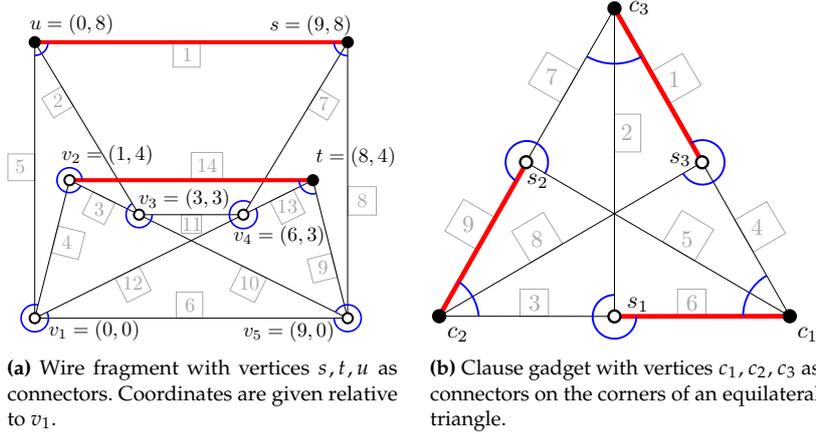


Figure 2.10: Blue arcs indicate the Λ -cone of each vertex. Red edges are candidates for the first or last scanned edge in a Λ -cover. Gray edge labels indicate a scan order of a Λ -cover.

For a variable $x \in X$, the variable gadget G_x consists of a chain of wire fragments, as depicted in Figure 2.11. Denoting the number of occurrences of x in I by k , we create $2k$ copies of the wire fragment G_f^i with vertices s_i, t_i, u_i . Rotate the wire fragments with even index by 180° . To combine the wire fragments, we identify the vertices s_i and s_{i-1} for even i and the vertices u_i and u_{i-1} for odd i . We define $V_x^c := \{t_{2i} \mid i = 1, \dots, k\}$ as the *connector vertices* of the variable gadget.

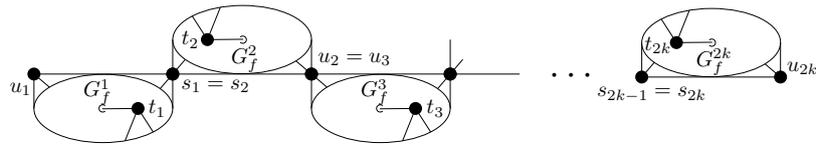


Figure 2.11: The variable gadget G_x of a variable $x \in X$ consists of $2k$ wire fragments, with every second copy rotated by 180° .

Lemma 2.3.6 *The variable gadget G_x has the following properties.*

1. In every Λ -cover of G_x , all vertices in V_x^c rotate in the same direction.
2. There exists a Λ -cover of G_x .

Proof. Suppose we have a Λ -cover of G_x . By Lemma 2.3.5, u_i and t_i rotate in the same direction, while s_i rotates in the opposite direction. Because the wire fragments are all connected at vertices with identical properties, all copies rotate in the same direction across all wire fragments; in particular, the vertices t_i rotate in the same direction.

For each wire fragment G_f^i in G_x , there exists a scan cover that is a Λ -cover for the fragment (Lemma 2.3.5). Concatenating the schedules from G_f^1 to G_f^k in this order yields a schedule for G_x . This results in a Λ -cover for G_x , as vertices used in two fragments rotate in the same direction and have two Λ -cones, one for each scan order of the two fragments. \square

For the construction of the clause gadget G_C of $C \in \mathcal{C}$, see Figure 2.10b. We place the *connector vertices* c_1, c_2, c_3 at the corners of an equilateral triangle, the vertices s_1, s_2, s_3 on the midpoints of the sides as illustrated, and insert the edges $c_i s_j$ for all $i, j \in \{1, 2, 3\}$. Using Observation 2.3.4, we ensure that the Λ -cones of s_1, s_2 , and s_3 correspond to the blue arcs in the figure. Note that a small perturbation suffices to obtain rational coordinates and does not harm the construction.

Lemma 2.3.7 *The clause gadget G_c has the following properties.*

1. *In every Λ -cover of G_c , not all connector vertices in G_c rotate in the same direction.*
2. *For each assignment of directions to connector vertices in G_c that does not assign them all the same direction, there exists a Λ -cover of G_c , such that every connector vertex rotates in its assigned direction.*

Proof. Consider a Λ -cover S of G_c . Note that a connector vertex c_i turns clockwise in S if and only if it scans the edge $s_i c_i$ first; this edge is highlighted red in Figure 2.10b.

By Observation 2.3.3, the edges $s_i c_i$ are the unique candidates for the first or last edge scanned in S . Consequently, at least one connector vertex turns clockwise, and at least one turns counterclockwise.

What remains to be shown is that for all configurations of not all equal rotations, there exists a scan order that covers the minimum angle. The order $c_3 s_3, c_3 s_1, c_2 s_1, c_1 s_3, c_1 s_2, c_1 s_1, c_3 s_2, c_2 s_3, c_2 s_2$ is minimal and has c_3 clockwise and c_1, c_2 counterclockwise. Reversing this order is also minimal and has c_1, c_2 clockwise and c_3 counterclockwise. Up to symmetry, these are all possible configurations in which c_1, c_2, c_3 do not all rotate in the same direction. \square

A wire gadget G_w consists of a chain of 18 wire fragments G_f^i such that two connectors on the ends of the chain differ in angle by θ . Observe that the angle between the bisectors of the maximum angles of vertices u_i and s_i is 90° . The construction consists of five parts that each will rotate the chain at an angle of $\theta/5$. (We use five parts to ensure the angle $90^\circ - \theta/5$ is not too small, which we need for Corollary 2.3.10) We connect the first four fragments, such that $u_2 = s_1, s_3 = s_2$, and $s_4 = u_3$ (Figure 2.12). We match the bisectors of all these connections, except for the connection between G_f^1 and G_f^2 , which is $90^\circ - \theta/5$. This is repeated four more times. The final part has only two wire fragments with an angle $90^\circ - \theta/5$ between them.

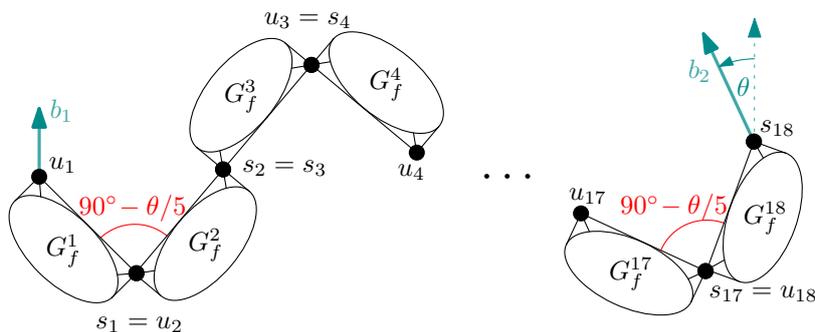


Figure 2.12.: The wire gadget consists of a chain of 18 wire fragments G_f^i . Red arcs indicate angles that vary such that the angle of b_1 and b_2 is equal to θ .

Lemma 2.3.8 *Let b_1 and b_2 be the bisectors of the outer cones of u_1 and s_{18} in the wire gadget G_w , respectively (see Figure 2.12). For any angle θ , the wire gadget G_w can be constructed such that the (counterclockwise) angle between b_1 and b_2 is θ and fulfills the following properties.*

1. *In every Λ -cover of G_w , both connector vertices u_1 and s_{18} in G_w*

rotate in the same direction.
 2. There exists a Λ -cover of G_w .

Proof. By construction, the angle between the bisectors of the outer cones of u_{4i+1} and s_{4i+2} is $\theta/5$, and the angle between the bisectors of the outer cones of s_{4i+2} and $s_{4(i+1)+1}$ is 0. Therefore, the angle between the bisectors of the outer cones of u_1 and s_{18} is $5 \cdot \theta/5 = \theta$.

Consider the order of the connector vertices of G_w , starting at u_1 , and denote the i -th vertex by c_i . Suppose we have a Λ -cover of G_w . Any pair of connector vertices in the chain that belong to the same wire fragment rotates in the opposite direction (Lemma 2.3.5.1). Therefore, connector vertices with an odd number of connector vertices between them on the chain rotate in the same direction. Therefore, u_1 and u_{18} rotate in the same direction.

We now give a schedule in which u_1 and s_{18} rotate clockwise. By Lemma 2.3.5.2, there exists a Λ -cover schedule relative to each wire fragment, such that c_i rotates clockwise if and only if i is odd. Concatenating the schedules from G_f^{18} to G_f^1 in this order yields a schedule for G_w . The vertices internal to the wire fragments have the same angles as in the earlier cover. Each connector vertex that is involved in one of the $90^\circ - \theta/5$ angles rotates in counterclockwise direction θ and has the angle of size $90^\circ - \theta/5$ inside the minimum cone, so the concatenation of the schedules covers this vertex with a minimum cone. Each connector vertex that is not involved in one of the $90^\circ - \theta/5$ angles has two Λ -cones, so these vertices can be scanned in a Λ -cover for any order of the gadgets. \square

Finally, to construct G_I , we first place corresponding variable and clause gadgets in the plane. We place the gadgets in a row with all clauses to the right of all variables. For a variable that occurs in a clause, we connect their gadgets by a wire gadget. To this end, we first identify a connector vertex $c := t_{2i}$ of the variable G_x and the connector vertex $\bar{c} := u_1$ of the wire gadget G_w . We rotate the wire such that the bisector b of the Λ -cone of c in G_x and the bisector \bar{b} of the outer cone of \bar{c} in G_w are equal. This ensures that the identified vertex $c = \bar{c}$ has two Λ -cones of size $\Lambda(c)$ (Figure 2.13).

By Lemma 2.3.8 we can choose the bisector of s_{18} corresponding to the connector vertex c' of the clause. With the clauses sufficiently far to the right, we can write $c' - s_{18}$ as linear combination of $s_1 - u_1$ and $s_2 - u_2$ with positive coefficients. Therefore, we can scale G_f^1 and G_f^2 to move s_{18} to c' . As above, we get two Λ -cones of the same size.

We are now ready to prove that MSC-TE and MSC-BE are NP-hard.

Theorem 2.3.9 *MSC-TE and MSC-BE in 2D are NP-hard, even for bipartite graphs.*

Proof. Given an instance $I = (X, C)$ of MNAE3SAT, we construct G_I with gadgets and auxiliary edges satisfying Lemmas 2.3.6 to 2.3.8. Observe that G_I is bipartite, because all gadgets are bipartite and the identified connector vertices can all be assigned the same color. Additionally, all vertices in G_I have the same $\Lambda(v)$ by applying Observation 2.3.4. We

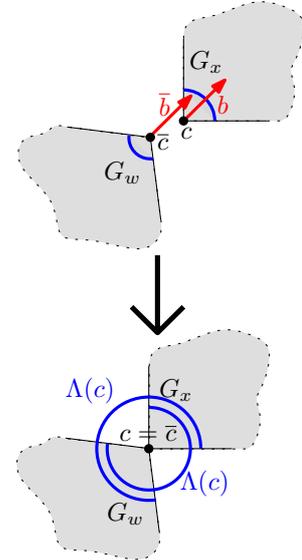


Figure 2.13.: Connecting the gadget G_w to G_x such that we have two possible Λ -cones.

show that I is satisfiable if and only if G_I has a Λ -cover. Because any optimal solution to MSC-TE and MSC-BE is a Λ -cover, if a Λ -cover exists, this implies that both problems are NP-hard.

Suppose I is satisfiable. Then there exists a variable assignment ϕ such that in no clause all values are equal. We construct a Λ -cover for G_I by specifying a scan order. We start with the variable gadgets. For each $x \in X$, scan all edges of G_x with a Λ -cover, as follows. If $\phi(x) = 1$, the connector vertices rotate clockwise, otherwise counterclockwise. By Lemma 2.3.6, such a scan order exists. Next, we scan all the wire gadgets G_w such that both connector vertices of G_w rotate in the same direction as the one attached to the vertex gadget did in the vertex gadget. By Lemma 2.3.8, such a scan order exists. Finally, scan the clause gadgets G_c . Each connector vertex already rotated (counter-) clockwise, depending on $\phi(x)$ of the attached variable x . Because ϕ is a valid (non-equal) assignment, no three connector vertices in a clause gadget rotate in the same direction. Thus, by Lemma 2.3.7 a Λ -cover of this clause gadget exists. Therefore, all edges can be scanned by a Λ -cover.

It remains to show the converse. Let S be a Λ -cover of G_I . We define $\varphi(v) = 1$ if v rotates clockwise over its Λ -cone, and $\varphi(v) = 0$ otherwise. We claim that if x_c is a connector vertex in gadget G_x , $\phi(x) = \varphi(x_c)$ yields a satisfying truth assignment to I . By Lemma 2.3.6, in any Λ -cover of G_I , all $\varphi(x_c)$ are equal for all connector vertices in G_x , thus $\phi(x)$ is well defined. By Lemma 2.3.8, in any Λ -cover, both connector vertices in gadget G_w rotate in the same direction. Finally, by Lemma 2.3.7, no clause gadget G_c contains three connector vertices rotating in the same direction in a Λ -cover. Therefore, $\phi(x)$ is a valid (non-equal) assignment of I , i.e., I is satisfiable. Because the reduction from I to G_I can be done in time polynomial in the number of clauses and variables, this concludes the proof. \square

Corollary 2.3.10 *MSC-BE in 2D is NP-hard to approximate within a factor of 1.04, even for bipartite graphs.*

Proof. Given an instance I of MNAE3SAT, consider the graph G_I constructed in the proof of Theorem 2.3.9. Recall that by Observation 2.3.4, we may assume that the angle of the Λ -cones coincide for all vertices. Define $\theta_{\max} := 360^\circ - \Lambda(v)$ for some vertex v . Let θ_{\min} denote the minimal angle over all incident pairs of edges, for which both vertices have degree greater than one.

In a scan cover S for G_I that is not a Λ -cover, there exists a vertex v with a total rotation angle exceeding $\Lambda(v)$. Consequently, either v scans its edges in a unidirectional rotation including the angle of size θ_{\max} (while possibly not scanning a smaller angle ε), or some angle between two edges at v is covered at least twice. We may assume that both vertices of those two edges have degree > 1 , because edges with a vertex of degree 1 can be reordered freely to get an unidirectional rotation. Therefore, this angle has size at least θ_{\min} .

Thus the objective value of S is at least $\min(360^\circ - \varepsilon, 360^\circ - \theta_{\max} + \theta_{\min})$, while the cost of a Λ -cover is $360^\circ - \theta_{\max}$. Let α be the ratio between these two quantities. To prove the corollary, it is sufficient to prove $\alpha > 1.04$.

By checking all gadgets, we observe that $\theta_{\max} = \arctan(1/2)$ (the angle outside the Λ -cone of v_5 in the wire fragment, see Figure 2.10a) and $\theta_{\min} \geq \arctan(1/4)$ (which is also realized at v_5). The angle ε can be chosen smaller than any given constant by adding additional edges. This results in $\alpha \geq \frac{360^\circ - \arctan(1/2) + \arctan(1/4)}{360^\circ - \arctan(1/2)} \approx 1.042$. \square

2.4. Approximations in 2D

In the previous section, we saw that already bipartite graphs are NP-hard to approximate within a specific factor for two-dimensional point sets. We now show that we can utilize geometric properties to obtain constant factor approximations for all objectives. For MSC-MS, we even get constant factor approximations for any graph where we can obtain a reasonably good coloring. Otherwise, we can provide approximation factors based on the chromatic number for all three objectives.

2.4.1. Bipartite Graphs in 2D

Conversely, we give absolute and relative performance guarantees for bipartite graphs in 2D.

Theorem 2.4.1 *Every bipartite instance $(P_1 \cup P_2, E)$ of MSC-MS has a scan cover with makespan 360° . Moreover, if P_1 and P_2 can be separated by a line, there exists a scan cover with makespan 180° .*

Proof. We show that the following strategy yields a scan cover of time 360° : All points turn in clockwise direction, with the points in P_1 starting with heading north and the points in P_2 with heading south; see Figure 2.14a for an example. Note that the connecting line between any point $p_1 \in P_1$ and any point $p_2 \in P_2$ forms alternate angles with the parallel vertical lines through p_1 and p_2 , so both face each other when reaching this angle during their rotation; see Figure 2.14b. In the case of separated point sets, a rotation of 180° suffices to sweep the other set, as illustrated in Figure 2.14c. \square

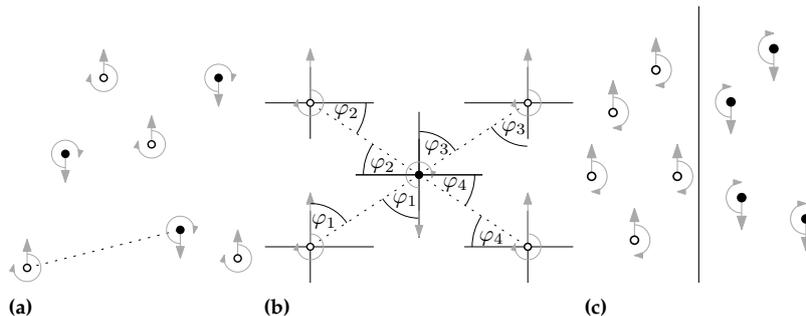
This result can directly be extended to arbitrary graphs.

Corollary 2.4.2 *Every instance (P, E) of MSC-MS has a scan cover with makespan $\lceil \log_2 |P| \rceil \cdot 360^\circ$.*

Proof. We can recursively partition the graph in the middle with a straight line $\lceil \log_2 |P| \rceil$ times and apply Theorem 2.4.1. To concatenate the $\lceil \log_2 |P| \rceil$ scan covers, we need at most an additional 180° per scan cover. \square

Theorem 2.4.1 yields an *absolute* bound for bipartite graphs. Now we give a constant-factor approximation even for small optimal values.

Figure 2.14.: (a) The vertices in P_1 and P_2 rotate clockwise and start by heading north and south, respectively. (b) Due to alternate angles, vertices of different parts of the vertex partition face each other at the same time. (c) When P_1 and P_2 are separated by a line, a scan time of 180° suffices.



Theorem 2.4.3 *There is a 4.5-approximation for MSC-MS for bipartite graphs $G = (P_1 \cup P_2, E)$ in 2D.*

Proof. Consider an instance I of MSC-MS in 2D and let Λ denote the minimum angle such that for every vertex some Λ -cone contains all its edges, i.e., $\Lambda = \max_{p \in P} \Lambda(v)$ with $P = P_1 \cup P_2$. Clearly, Λ is a lower bound on the value OPT of a minimum scan cover of I .

We use one of two strategies depending on Λ . If $\Lambda \geq 90^\circ$, we use the strategy of Theorem 2.4.1 which yields a scan cover of at most 360° and hence a 4-approximation.

If $\Lambda < 90^\circ$, we use an adaptive strategy as follows. For each vertex, we partition the set of headings $[0, 360^\circ)$ into $2s$ sectors of size $\Lambda' = 360^\circ/2s$, see Figure 2.15.

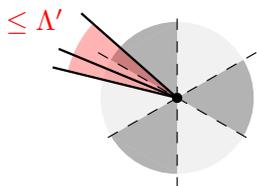


Figure 2.15.: Dividing headings into $2s$ sectors with angle $\Lambda' = 360^\circ/2s$ by inserting s lines. For every vertex, the incident edges lie in at most two adjacent sectors of size Λ' , because $\Lambda \leq \Lambda'$.

We choose s maximal (and, thus, Λ' minimal) such that $\Lambda' \geq \Lambda$. This implies that the edges of every vertex are contained in at most two adjacent sectors, see Figure 2.15. Note also that $\Lambda' < 3/2\Lambda$, because $\Lambda > 360^\circ/2(s+1)$ and $s \geq 2$.

Let the sectors be $c_i = [i \cdot \Lambda', (i + 1) \cdot \Lambda')$, for $i = 0, \dots, 2s - 1$. Moreover, $c'_i := c_{i+s \bmod 2s}$ is the sector *opposite* of c_i . Note that an edge $e = vw$ is in the sector c_i of v if and only if e is in the opposite sector c'_i of w , see Figure 2.16.

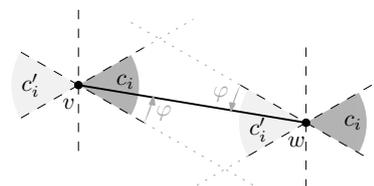


Figure 2.16.: An edge $e = vw$ that lies in c_i for v , lies in $c'_i = c_{i+s}$ for w . If v scans c_i and w scans c'_i (both counterclockwise), they scan e at the same time φ due to the alternate angles in the parallelogram.

Let C_{even} be the set of sectors with even indices, C_{odd} the one with odd indices, and C'_{even} and C'_{odd} the set of opposite sectors, respectively. Because the incident edges of each vertex are contained in at most two adjacent sectors, every vertex has edges in (at most) one sector of C_{even} and one sector of C_{odd} .

This allows the following strategy. In the first phase, the vertices in P_1 scan the sector with edges in C_{even} in clockwise direction, while the vertices in P_2 scan the sector in C'_{even} . In the second phase, vertices in P_1 scan the sector in C_{odd} in counterclockwise direction, while the vertices in P_2 scan the sector in C'_{odd} .

As in Theorem 2.4.1, every edge is scanned in the first or second scan phase due to the alternate angles. Clearly, each scan phase needs Λ' . Between the two scan phases, every vertex v needs to turn to change its heading from the end heading of the first scan phase to the start heading of the second. Because both sectors of v are incident and due to the reversed direction, the turning angle is at most Λ' ; in particular, either the end heading of the first sector is contained in the boundary

of the second sector or the two start headings of both phases coincide. Figure 2.17 depicts an example scan cover. The resulting scan time is $3\Lambda' \leq 3 \cdot 3/2 \cdot \Lambda \leq 4.5 \cdot \text{OPT}$. \square

Next, we complement the 4.5-approximation algorithm of Theorem 2.4.3 for MSC-MS in bipartite graphs in the plane by presenting an approximation algorithm for both remaining objectives.

Theorem 2.4.4 *There exists a 2-approximation algorithm for MSC-BE and MSC-TE for each bipartite graph $G = (P_1 \cup P_2, E)$ embedded in the plane.*

Proof. Defining $P := P_1 \cup P_2$, the values $\max_{p \in P} \Lambda(p)$ and $\sum_{p \in P} \Lambda(p)$ are clearly lower bounds on the value of a scan cover minimizing MSC-BE and MSC-TE, respectively.

We use the same strategy based on alternating angles as in Theorem 2.4.1: Defining start headings $r(p, 0) := 0^\circ$ for $p \in P_1$ and $r(p, 0) := 180^\circ$ for $p \in P_2$, the clockwise rotation scheme induces a scan cover S by defining the scan time $S(e)$ of edge e as the time when its two vertices face each other.

We now show that in the rotation scheme induced by S , i.e., every vertex p starts to head toward its edge first scanned in S and then follows the order on $E(p)$ defined by S , the total rotation angle of each vertex p is at most $2\Lambda(p)$. To this end, we consider three types of vertices; for an illustration consider Figures 2.18 to 2.20.

Case (a): $r(p, 0)$ lies outside the Λ -cone of p . Then all edges of p are scanned by a clockwise rotation, one after the other. Hence, p has a total rotation angle of $\Lambda(p)$.

Case (b): $r(p, 0)$ lies inside the Λ -cone of p and $\Lambda(p) \geq 180^\circ$. Going over all edges clockwise takes at most a full rotation of $360^\circ \leq 2\Lambda(p)$.

Case (c): $r(p, 0)$ lies inside the Λ -cone of p and $\Lambda(p) < 180^\circ$. Let e_1 and e_2 denote the bounding edges of the Λ -cone such that $S(e_1) \leq S(e_2)$. By definition, the minimal angle of e_1 and e_2 is $\Lambda(p) < 180^\circ$. Splitting the Λ -cone of p into two halves at $r(p, 0)$, p scans the edges in each half in clockwise direction, rotating an angle of $\Lambda(p)$ counterclockwise between e_1 and e_2 . It follows that the total rotation angle of p is at most $2\Lambda(p)$.

As the total rotation angle is at most $2\Lambda(p)$ for each vertex p , MSC-BE and MSC-TE are upper bounded by $2 \cdot \max_{p \in P} \Lambda(p)$ and $2 \cdot \sum_{p \in P} \Lambda(p)$. Together with the lower bounds provided above, this shows that this scan cover is a 2-approximation for *either* objective. \square

Corollary 2.4.5 *Let $G = (P_1 \cup P_2, E)$ be a bipartite graph embedded in the plane such that the points of P_1 and P_2 can be separated by a line. Then an optimal MSC-BE and MSC-TE of G can be found in polynomial time.*

Proof. We follow the same technique as in the proof of Theorem 2.4.4. We may assume without loss of generality that the separating line is vertical and that the points of P_1 lie left of the line. Then, with the above definitions, every vertex is in case (a), i.e., the total rotation angle for each

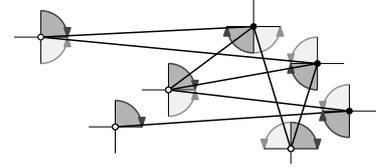


Figure 2.17.: An example with $\Lambda' = 90^\circ$. P_1 and P_2 are indicated by white and black vertices, respectively. The dark sectors are scanned in the first scan phase; the light sectors in the second. Each scan phase and the turning phase costs Λ' .



Figure 2.18.: Case (a).



Figure 2.19.: Case (b).

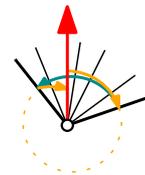


Figure 2.20.: Case (c).

vertex p is $\Lambda(p)$. Consequently, the resulting scan cover is optimal for both MSC-BE and MSC-TE. \square

2.4.2. Graphs with Bounded Chromatic Number

Like in 1D, the makespan of a minimum scan cover in 2D has a strong relation to the chromatic number. More specifically, we show that the optimal scan time lies in $\Theta(\log_2 \chi(G))$ and that for a given coloring of the graph G with $\chi(G)^c$ colors, we can provide an $O(c)$ -approximation.

Lemma 2.4.6 *Let G be a graph embedded on a points set $P \subset \mathbb{R}^d, d > 1$. If there exists a scan cover of makespan $T > 0$, then G has a cut cover of size $d \cdot \lceil T/90^\circ \rceil$, i.e., $c(G) \leq d \cdot \lceil T/90^\circ \rceil$.*

Proof. Partition the scan cover into $\lceil T/90^\circ \rceil$ intervals of length at most 90° . For each interval i , we consider the set of edges that are scanned within this interval, inducing a graph G_i . We show that each G_i is 2^d -partite. Because $c(G_i) = \lceil \log_2(\chi(G_i)) \rceil \leq \lceil \log_2(2^d) \rceil = d$ and $c(G) \leq \sum_i c(G_i) \leq \lceil T/90^\circ \rceil \cdot d$, this implies the claim. We first consider the case $d = 2$. We classify the points of P into four sets, depending on their turning behavior within the interval i . Each point has a quadrant $[0, 90^\circ)$, $[90^\circ, 180^\circ)$, $[180^\circ, 270^\circ)$, or $[270^\circ, 360^\circ)$ to which it is heading at the time 45° ; we assign each point this quadrant. Note that every point can only leave its assigned quadrant by less than $\pm 45^\circ$. Two points that are assigned to the same quadrant are independent in G_i : When their edge is scanned, the headings of the two points have to be opposite, i.e., they differ by exactly 180° . Thus, the only case in which two point headings could differ by 180° is if one leaves its quadrant by 45° in clockwise and the other by 45° in counterclockwise direction. However, in this case, the points would not have been assigned to the same half-open sector. For an illustration consider Figure 2.21.

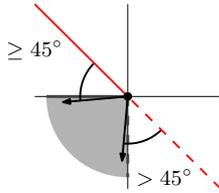


Figure 2.21.: Every vertex is assigned to the (gray) sector to which it is heading at time 45° . The boundary of the reachable headings within 90° is shown in red. Since the sector is half-open, two vertices assigned to the same sector cannot reach opposing headings and thus cannot scan their edge.

For $d \geq 3$, the idea is analogous. To simplify the argument we choose a coordinate system, i.e., an orthonormal basis (ONB), such that, at time 45° , no point heads in a direction that lies on a lower-dimensional subspace spanned by the basis vectors. Let B denote the set of all potential basis vectors in \mathbb{R}^d , i.e., $B = \mathbb{R}^d$. For every point, we delete the line spanned by its heading h at time 45° and the $(d - 1)$ -dimensional subspace orthogonal to h from B . The remaining set B' is a d -dimensional space minus a finite number of lower-dimensional subspaces. It follows by induction that B' contains an ONB.

The points of P are partitioned into 2^d different sets, depending on the orthant in which they are contained at time 45° . Note that if two point headings of the same orthant differ by 180° , their angle difference at time 45° is 90° , i.e., they lie on a lower dimensional subspace spanned by the basis vectors. This is a contradiction to the choice of the ONB. \square

Because $c(G) = \lceil \log_2 \chi(G) \rceil$, Lemma 2.4.6 has the following implication.

Lemma 2.4.7 *Let I be an instance of MSC-MS in \mathbb{R}^d with graph G . Then every scan cover has a makespan T of at least $\Omega(\log_2 \chi(G))$. More precisely, it holds that $T \geq 1/d(\lceil \log_2 \chi(G) \rceil - d) \cdot 90^\circ$.*

Proof. Lemma 2.4.6 implies that $\lceil \log_2 \chi(G) \rceil \leq d \lceil T/90^\circ \rceil$ because $c(G) = \lceil \log_2 \chi(G) \rceil$. In particular, it holds that $\lceil \log_2 \chi(G) \rceil \leq d(T/90^\circ + 1)$. This is equivalent to $1/d(\lceil \log_2 \chi(G) \rceil - d) \cdot 90^\circ \leq T$. \square

These insights have the following implications.

Theorem 2.4.8 *For a k -colored graph G embedded in 2D with $k \leq \chi(G)^f$ for some function f , there exists an $O(f)$ -approximation for MSC-MS.*

Proof. Partition G into $\lceil \log_2 k \rceil$ bipartite graphs G_i . By Theorem 2.4.3, each G_i can be scanned in time $\beta \cdot OPT_i$, with $\beta = 4.5$ and OPT_i denoting the optimum of the instance induced by G_i . Clearly, $OPT_i \leq OPT$, and turning from the last scan of one bipartite graph to the next takes at most a time of OPT . Hence, this scan cover needs a time of at most $(\beta + 1)OPT \lceil \log_2 k \rceil$. If $\chi(G) \leq 4$, then

$$\begin{aligned} O(\lceil \log_2 k \rceil) &\subseteq O(\lceil f \cdot \log_2(\chi(G)) \rceil) \subseteq O(\lceil f \cdot \log_2(4) \rceil) \\ &\subseteq O(\lceil 2f \rceil) = O(f). \end{aligned}$$

If $\chi(G) \geq 5$, then $\log_2(\chi(G)) > 0$ and Lemma 2.4.7 ensures that $OPT \in \Omega(\log_2(\chi(G)))$. Therefore, the performance guarantee is in

$$O\left(\frac{\log_2 k}{\log_2(\chi(G))}\right) = O\left(\frac{f \cdot \log_2(\chi(G))}{\log_2(\chi(G))}\right) = O(f).$$

\square

As a direct implication of Theorem 2.4.8, we get a spectrum of approximation algorithms for interesting special cases.

Corollary 2.4.9 *MSC-MS in 2D allows the following approximation factors.*

1. $O(\log_2 n)$ for every graph G with n vertices.
Furthermore, the minimum scan time lies in $\Theta(\log_2 \chi(G))$.
2. $O(1)$ for planar graphs.
3. $O(\log_2 d)$ for d -degenerate graphs.
4. $O(1)$ for graphs of bounded treewidth.
5. $O(1)$ for complete graphs.

The following bound shows a refined approximation for complete graphs.

Corollary 2.4.10 *Consider the MSC-MS for complete graphs with n vertices in 2D. There is a C -approximation algorithm with $C \rightarrow 6$ for $n \rightarrow \infty$.*

Proof. We may assume without loss of generality that $n > 4$. The minimum scan time is at least $(\lceil \log_2(n) \rceil - 2) \cdot 45^\circ > 0$ due to Lemma 2.4.7. For the upper bound, we partition the point set recursively into $\lceil \log_2(n) \rceil$ bipartite graphs by lines (alternating horizontal and vertical). Hence, Theorem 2.4.1 allows us to scan each bipartite graph within 180° . The transition between two scan phases is at most 90° . Therefore, the scan time is upper bounded by $\lceil \log_2(n) \rceil \cdot 180^\circ + (\lceil \log_2(n) \rceil - 1) \cdot 90^\circ$. This yields a performance guarantee of

$$\frac{270^\circ(\lceil \log_2(n) \rceil - 2) + 450^\circ}{45^\circ(\lceil \log_2(n) \rceil - 2)} = 6 + \frac{10}{(\lceil \log_2(n) \rceil - 2)}. \quad \square$$

The factor in Corollary 2.4.10 is $C \leq 8$ when $n \geq 2^7$ and $C \leq 7$ when $n \geq 2^{12}$.

The insights from Theorem 2.4.4 also yield an approximation algorithm for k -colored graphs for MSC-TE and MSC-BE.

Corollary 2.4.11 *For MSC-TE and MSC-BE of k -colored graphs embedded in 2D, there exists an $O(\log(k))$ -approximation.*

Proof. The edges of a k -colored graph G can be covered by $\lceil \log_2(k) \rceil$ bipartite graphs G_i [38]. For each G_i , we use the 2-approximation of Theorem 2.4.4. Clearly, for both objectives, the optimal scan time of G is lower bounded by the optimal scan time of every subgraph. Consequently, scanning all G_i takes at most $\sum_i 2 \cdot OPT(G_i) \leq 2\lceil \log_2(k) \rceil \cdot OPT(G)$, where OPT denotes the optimum scan time for the respective objective. For adjusting the headings between the scan covers of the bipartite graphs, we need $(\lceil \log_2(k) \rceil - 1)$ transition phases each of which needs at most $OPT(G)$. Hence, the total scan time is upper bounded by $(3\lceil \log_2(k) \rceil - 1) \cdot OPT(G)$. \square

Future Work 2.2.

Can we still provide constant-factor approximations if every scan takes some time? This would force us to interrupt the sweeps and make the efficient synchronisation to match the alternating angles more challenging. Another question is if we can preserve the constant approximation factor if we have non-uniform rotation speeds. While this can allow us to perform many scans earlier, the slowest satellite could still dominate the makespan.

2.5. 3D and Abstract Metrics

In the following, we observe that Δ MSC-MS generalizes the Path-TSP on a star graph. Because all objectives are equal on stars, this also carries over to the other two objectives.

Observation 2.5.1 *Let $G = (V, E)$ be a star on $n + 1$ vertices with center v*

and α a metric transition cost function on $E \times E$. Then, an Δ MSC of (G, α) for any of the three objectives corresponds to a TSP-path of the complete graph on $V \setminus \{v\}$ with metric cost $c(u_1, u_2) = \alpha(vu_1, vu_2)$ and vice versa.

Observation 2.5.1 has two immediate consequences. Firstly, because the metric Path-TSP is NP-hard, it follows that the abstract version is already hard on stars.

Observation 2.5.2 Δ MSC is NP-hard for all three objectives, even for stars.

Secondly, a $3/2$ -approximation for metric Path-TSP [52, 53] can be applied. We also note that a recent result provides an improved approximation factor of $3/2 - 10^{-36}$ [54].

Observation 2.5.3 There exists a $3/2$ -approximation for Δ MSC on stars.

In contrast to 1D and 2D, we show that the chromatic number does not provide an upper bound for MSC-MS in 3D and Δ MSC-MS.

Observation 2.5.4 There are instances of MSC-MS in 3D for which the graph G has n vertices, $\chi(G) = 2$ and every scan cover has a makespan in $\Omega(\sqrt{n})$. There are instances of MSC-MS in \mathbb{R}^d for which the graph G has n vertices, $n \in O(d)$, $\chi(G) = 2$ and every scan cover has a makespan in $\Omega(n)$.

Proof. For the first claim, we consider a geodesic triangular grid with n vertices on a sphere and embed a star graph such that its leaves are grid points and the center of the star lies in the center of the sphere.

Clearly, the makespan is lower bounded by $(n - 1) \cdot \ell$, where ℓ is the minimum angle between all edge pairs of the central vertex. We prove our bound by refining the resolution of the grid, i.e., we iteratively insert one vertex per edge as illustrated in Figure 2.22: Because a triangulation with n vertices has $3n - 6$ edges, the number of vertices in the refined grid is $n + 3n - 6 = 4n - 6$, i.e., one refining step roughly quadruples the number of vertices. Moreover, the minimum angle between any two consecutive edges of the central vertex of the star is approximately cut in half during a refining step. Due to the curvature, the edges between the new vertices can actually be slightly less than half of the minimum angle; for $n \rightarrow \infty$, the difference becomes infinitesimal small. Hence, G_i has $N \in \Theta(4^i n)$ vertices and a minimum angle $\beta \in \Theta(\ell/2^i)$ yielding a lower bound of $\Omega(N\beta) = \Omega((2^i n - 2^{-i}) \cdot \ell) \subseteq \Omega(2^i \sqrt{n}) = \Omega(\sqrt{N})$. Note that already the expected length of randomly distributed n vertices in a unit square is proportional to \sqrt{n} for large n [55].

For the second claim, we consider a star on $n + 1$ vertices for which each of the n leaves is placed on a different coordinate axis in \mathbb{R}^n . Therefore, the turn cost between any two edges is 90° and it takes $90^\circ(n - 1)$ to scan all edge of the graph. Thus, we have a lower bound of $\Omega(n)$. \square

The approximation technique for bipartite graphs in 2D relies on alternate angles and fails for 3D or Δ MSC-MS. Nevertheless, we provide a 2.5-approximation for trees based on Observation 2.5.1.

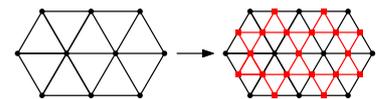


Figure 2.22.: A refining step of the geodesic grid.

Theorem 2.5.5 *There exists a 2.5-approximation for Δ MSC for trees.*

Proof. For the minimum energy and bottleneck energy, we only have to apply Observation 2.5.1 to every vertex. Because the graph is a tree, there cannot be any cyclic dependencies and every scan order is feasible.

For Δ MSC-MS, we have to be more careful: Let $I = (G, \alpha)$ be an instance of Δ MSC-MS for which G is a tree, and let OPT be the minimum scan time of I . For every vertex v , we approximate an ordering of minimum cost over all its incident edges E_v . Let $N(v)$ denote the set of neighbors of v . By Observation 2.5.1 such an ordering corresponds to a TSP-path. Consequently, we may use a $3/2$ -approximation algorithm for metric Path-TSP [52, 53]. Moreover, we enhance the edge ordering to a cyclic ordering by inserting an edge from the last to the first edge; because the cost function is metric, the cost of the additional edge is upper bounded by the minimum cost ordering of the incident edges. Therefore, the scan time ℓ_v of the computed cyclic edge ordering of v is at most $\ell_v \leq (3/2 + 1)OPT$.

We construct a scan cover as follows: Every vertex follows its cyclic edge ordering. The start headings of the vertices are chosen such that the scan time of each edge $e = uv$ is synchronized at the vertices u and v . To this end, we choose some vertex r as the root and denote the parent of each vertex v by $par(v)$ in the tree G with respect to the root r . We scan the edges of r according to the cyclic edge orderings by starting with any heading, see also Figure 2.23.

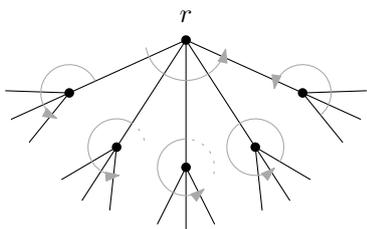


Figure 2.23: Root r can choose its schedule. The (cyclic) schedules of the children are synchronized with the timing of its parent. Because the graph is a tree, there are no cyclic dependencies.

We then determine the start headings by a tree traversal (such as DFS or BFS): Let v be a vertex whose start heading has to be determined, and assume the start heading of $u := par(v)$ is already fixed. When uv is scanned at time t for u , then the cyclic ordering of E_v is shifted, so that v sees uv also at time t . If this time lies between two scans, we simply start at the next incident edge and let the vertex wait for the appropriate time. Because all vertices start at the same time, the resulting scan cover has a scan time of at most $\max_v \ell_v \leq 2.5 \cdot OPT$. \square

Theorem 2.5.5 allows an approximation algorithm in terms of the arboricity of the underlying graph. Recall that the *arboricity* of a graph denotes the minimum number of forests into which its edges can be partitioned.

Theorem 2.5.6 *There exists a 3.5τ -approximation for the Δ MSC for graphs of arboricity τ .*

Proof. We compute a decomposition into τ forests in polynomial time [56]. To obtain a scan cover we use the approximation algorithm of Theorem 2.5.5 for each forest and concatenate the resulting scan covers in any order. Because the transition cost between any two forests is upper bounded by the minimum scan time/cost OPT , the resulting scan cover has time/cost of at most $(2.5 + 1)OPT \cdot \tau$. Consequently, we obtain a 3.5τ -approximation. \square

Future Work 2.3.

Can we find an approximation algorithm that does not depend on the arboricity or disprove its existence?

2.6. Computational Study

For our experimental evaluation, we considered two types of benchmark instances in 2D, which we call *random* and *celestial*. Random instances are generated by placing n points chosen uniformly at random from the unit square, with each edge chosen with probability p . Note that the visible area of a satellite constellation on the same altitude in low Earth orbit is fairly close to a set of co-planar points and hence the square (or plane) serves as a reasonable approximation. An example can be seen in Figure 2.24.

Celestial instances are inspired by real-world instances of satellites in a shared orbit, in which they maintain their relative positions while orbiting around a central body like Earth, as long as no explicit orbit-changing maneuvers are carried out. They are characterized by a set of points on a circle and a central circular obstacle. The points on the circle are chosen uniformly at random; an edge exists if and only if its vertices *see* each other, i.e., the edge does not intersect the central obstacle. An example can be seen in Figure 2.25

The distribution of the nearly 2000 instances with up to 800 edges used for our experiments can be seen in Figure 2.26.

All experiments were run on Intel Core i7-3770 with 3.4 GHz and 32 GB of RAM.

2.6.1. Optimal Solutions

We developed three mixed integer programs (MIPs) and two constraint programs (CPs) to solve instances to provable optimality. Note that not every program solves all three problems. An experimental evaluation is given at the end of this section. While we focus on two-dimensional geometric instances, all formulations are applicable to all metric cost functions.

Mixed Integer Program 1 (MSC-MS, MSC-TE, MSC-BE)

Our first MIP, denoted by MIP-1, uses two types of variables. The first type are real variables $t_e \geq 0$ for all $e \in E$, see Figure 2.27.

The second type are Boolean variables $x_{(e,e')}$ for all ordered edge pairs $(e, e') \in E^2$, see Figure 2.28.

In a computed solution, the variables t_e define a scan cover in which $S(e) := t_e$ and the value of $x_{(e,e')}$ corresponds to $v_S(e, e')$. Because $v_S(e, e') = 0$ if $|e \cap e'| \neq 1$, we directly set $x_{(e,e')} := 0$ in these cases. Consequently, the objective functions can be expressed by substitution of $S(e)$ with t_e and $v_S(e, e')$ with $x_{(e,e')}$. Note that a min-max objective can

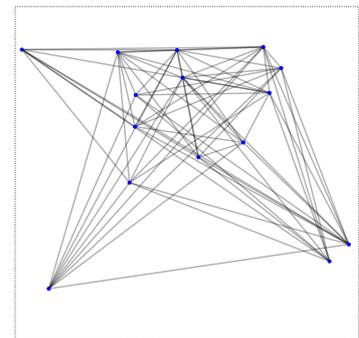


Figure 2.24.: A random instance with 15 vertices and 68 edges.

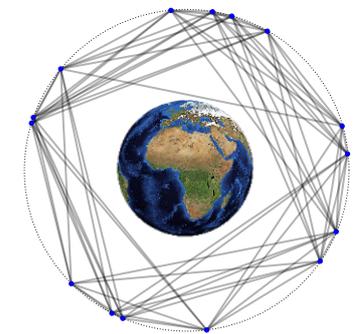


Figure 2.25.: A celestial instance with 15 vertices and 72 edges.

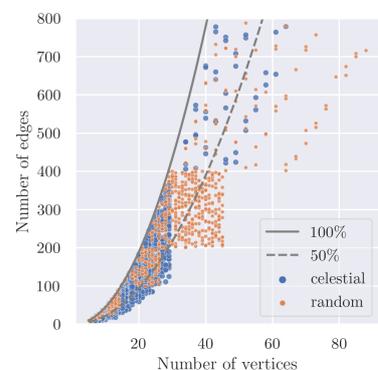


Figure 2.26.: Instance distribution of the nearly 2000 instances. The auxiliary lines indicate graphs with edge densities 50% (dashed) and 100% (solid) of complete graphs.

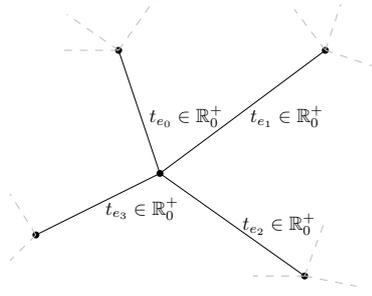


Figure 2.27.: Every edge e has a scan time variable $t_e \geq 0$.

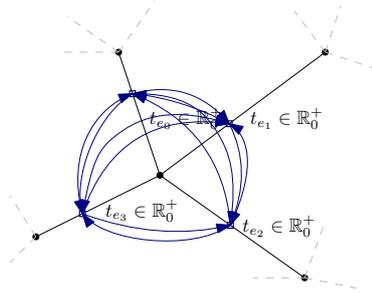


Figure 2.28.: We introduce variables $x_{(e,e')} \in \mathbb{B}$ for every rotation (blue) between scanning two edges e, e' .

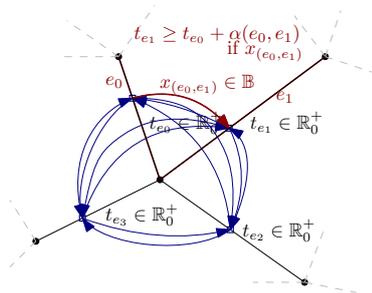


Figure 2.29.: When using a rotation, the necessary rotation time propagates.

be implemented by a single additional real variable and one additional constraint for each term in the objective.

We introduce a set of constraints to guarantee that the t -variables and the x -variables arise from a valid scan cover. Because the angle function α fulfills the triangle inequality, it suffices to ensure the time difference of the t -variables for all $x_{(e,e')} = 1$, see Figure 2.29.

We know that $M_1 := \lceil \log_2 n \rceil \cdot 360^\circ$ is an upper bound on the minimal makespan for a graph G in 2D with n vertices (Corollary 2.4.2). Moreover, a makespan of $M_2 := |E| \cdot 180^\circ$ allows to scan each edge individually, and thus an optimal scan cover of MSC-BE and MSC-TE can be realized in this makespan. Therefore, by inserting the correct M_i , we can enforce feasible scan times by using the Big-M method.

$$\forall v \in V, \forall (e, e') \in E(v) \times E(v), e \neq e': \quad t_{e'} \geq t_e + \alpha(e, e') - (1 - x_{(e,e')}) \cdot M_i. \quad (2.7)$$

This leaves us with ensuring that the x -variables correspond to a feasible scan cover. First, for every vertex v , an incident scanned edge e has at most one predecessor edge and one successor edge in the scan order.

$$\forall v \in V, e \in E(v): \quad \sum_{e' \in E(v), e' \neq e} x_{(e,e')} \leq 1 \quad \text{and} \quad \sum_{e' \in E(v), e' \neq e} x_{(e',e)} \leq 1 \quad (2.8)$$

Second, the total number of scanned edges at vertex v is $|E(v)|$, i.e., the number of consecutively scanned edge pairs, is $|E(v)| - 1$.

$$\forall v \in V: \quad \sum_{e, e' \in E(v) \times E(v), e \neq e'} x_{(e,e')} = |E(v)| - 1 \quad (2.9)$$

Together, Equations 2.8 and 2.9 enforce that every vertex has exactly one first and one last scanned edge in the induced scan order. Because Equation 2.7 enforces that the scan times obey the rotation times, there are no cycles in the sequence defined by x if all angles are positive. This fact is very similar to the Miller-Tucker-Zemlin formulation of the TSP [57]. In the presence of 0° -angles, we dynamically add the following constraint similar to the Dantzig formulation [58] to separate these cycles.

$$\forall v \in V, \forall S \subseteq E(v), S \neq \emptyset: \quad \sum_{e \in S, e' \in E(v) \setminus S} x_{(e,e')} + x_{(e',e)} \geq 1 \quad (2.10)$$

Mixed Integer Program 2 (MSC-MS)

The abstract definition of the MSC-MS can be directly implemented as a MIP, because absolute values can be implemented using a Boolean variable. Some modern solvers like Gurobi actually provide this functionality directly. Like for MIP-1 (Subsection 2.6.1), we have a real-valued variable $t_e \geq 0$ for each $e \in E$ that states its scan time. We try to keep the maximum value assigned to any $t_e, e \in E$ as low as possible. For every two incident edges vw and vu , we only have the constraint that t_{vw} and t_{vu} have to be at least the time apart that v needs to rotate between these

two. This results in the following MIP-2.

$$\min \quad \max_{e \in E} t_e \quad (2.11)$$

$$\text{s.t.} \quad |t_{vw} - t_{vu}| \geq \alpha(vu, vw) \quad \forall vw, vu \in E \quad (2.12)$$

$$t_e \geq 0 \quad \forall e \in E \quad (2.13)$$

The main difference to MIP-1 is that we do not keep a record of the actually performed rotations. As a consequence, MIP-2 can only be used for MSC-MS. However, on the positive side, we do not need to dynamically add additional cycle constraints.

Mixed Integer Program 3 (MSC-TE, MSC-BE)

The third MIP (defined by Equations 2.8 to 2.10 and 2.14), denoted by MIP-3, is a variant of MIP-1 (Subsection 2.6.1) in which the t -variables and the corresponding Big-M based constraint (Equation 2.7) are removed. As a consequence, we may use it for MSC-BE and MSC-TE, as they only need the x -variables.

It is possible that the scan orders at the individual vertices are cycle free, but that the overall schedule has a deadlock when the vertices wait for each other, see Figures 2.30a and 2.30b. We therefore prohibit directed cycles in the scan order defined by the x -variables (if not already separated by Equation 2.10) dynamically via callbacks for every newly found integral solution. Violated constraints can be found via a simple DFS search.

$$\forall k \in \mathbb{N}_{|V|}, \forall (e_0, e_1, \dots, e_{k-1}) \in E^k: \quad x_{(e_{k-1}, e_0)} + \sum_{i=0,1,\dots,k-2} x_{(e_i, e_{i+1})} \leq k-1 \quad (2.14)$$

Note that these cycles can also happen in MIP-1, but only with zero rotation costs between the involved edges. Thus, they are irrelevant for the solution, as all of these edges can be scanned at once.

Constraint Program 1 (MSC-MS)

Our first constraint program (denoted by CP-1) has the same formulation as MIP-2. The only difference between the CP version and the MIP version

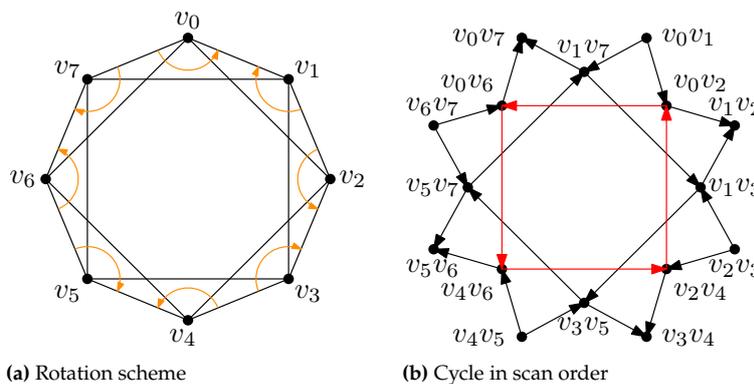


Figure 2.30.: A globally infeasible edge order fulfilling Equations 2.8 and 2.9, i.e., it is cycle-free at each vertex: (a) its rotations scheme (b) the resulting edge order that contains a cycle. An arc (e, e') in this graph corresponds to an $x_{(e, e')} = 1$.

lies in the employed solver. In particular, absolute values can be modeled directly.

Constraint Program 2 (MSC-TE, MSC-BE)

Our second constraint program (defined by Equations 2.8, 2.9 and 2.15), denoted by CP-2, is similar to MIP-3 described in Subsection 2.6.1. However, MIP-3 adds Equations 2.10 and 2.14 dynamically, which our CP does not support. Because adding all these constraints directly results in a prohibitively large formulation, we instead use a conditional variant of the Miller-Tucker-Zemlin [57] formulation to eliminate cycles in the scan order. Different from MIPs, we do not need the Big M method for CPs, but can implement conditional constraints directly. More precisely, we add the variables $o_e \in \mathbb{N}_{|E|}$, $e \in E$ that state the cycle-free scan order of the edges, which is enforced by the constraints

$$\forall (e, e') \in E \times E: \quad o_{e'} - o_e \geq 1 \quad \text{if } x_{(e,e')} = 1. \quad (2.15)$$

Experimental Evaluation of Exact Algorithms

We used *Gurobi* (v9.0.1) for solving the MIPs and *CP-SAT* of Google's *or-tools* (v7.7.7810) for solving the CPs. CP-SAT, which is based on a SAT solver, requires all coefficients and variables to be integral for computational efficiency. We therefore convert the floating point values to integral values including the first eight floating point digits (rounded, decimal). While this weakens the accuracy, we calculated a theoretical maximal deviation of less than 1×10^{-4} %, which we consider negligible and comparable to the accuracy of the MIP solver.

We considered all solvers for the three objectives on the two instance types described in the preliminaries. We evaluated how many instances of which size could still be solved to provable optimality within a time limit of 900 sec; see Figure 2.31. For MSC-MS, CP-1 has a clear lead, solving 50 % of the instances with $242 \pm 5\%$ edges for random instances, and $125 \pm 5\%$ edges for celestial instances. In our experiments, neither MIPs was able to solve any instance with more than 70 edges to provable optimality. For MSC-TE, MIP-1 and MIP-3 performed better than CP-2, but all solvers could barely solve instances with more than 30 edges. While MIP-1 has a more direct objective without auxiliary constraints and variables as needed for MSC-MS, its actual performance was slightly worse. For MSC-BE, CP-2 performed considerably better; for celestial instances, it can solve instances nearly twice as large ($\geq 50\%$ at $48 \pm 5\%$ edges) than the MIPs. Surprisingly, MIP-1 was slightly better than CP-2 for random instances, being able to solve 50 % of the instances with $61 \pm 5\%$ edges. Overall, CPs appear to be considerably more effective for this kind of problem than MIPs, and random instances show to be easier to solve than celestial ones.

2.6.2. Good Solutions: Approximations and Heuristics

For larger instances (beyond the size that was solvable to provable optimality), we developed additional methods based on approximation

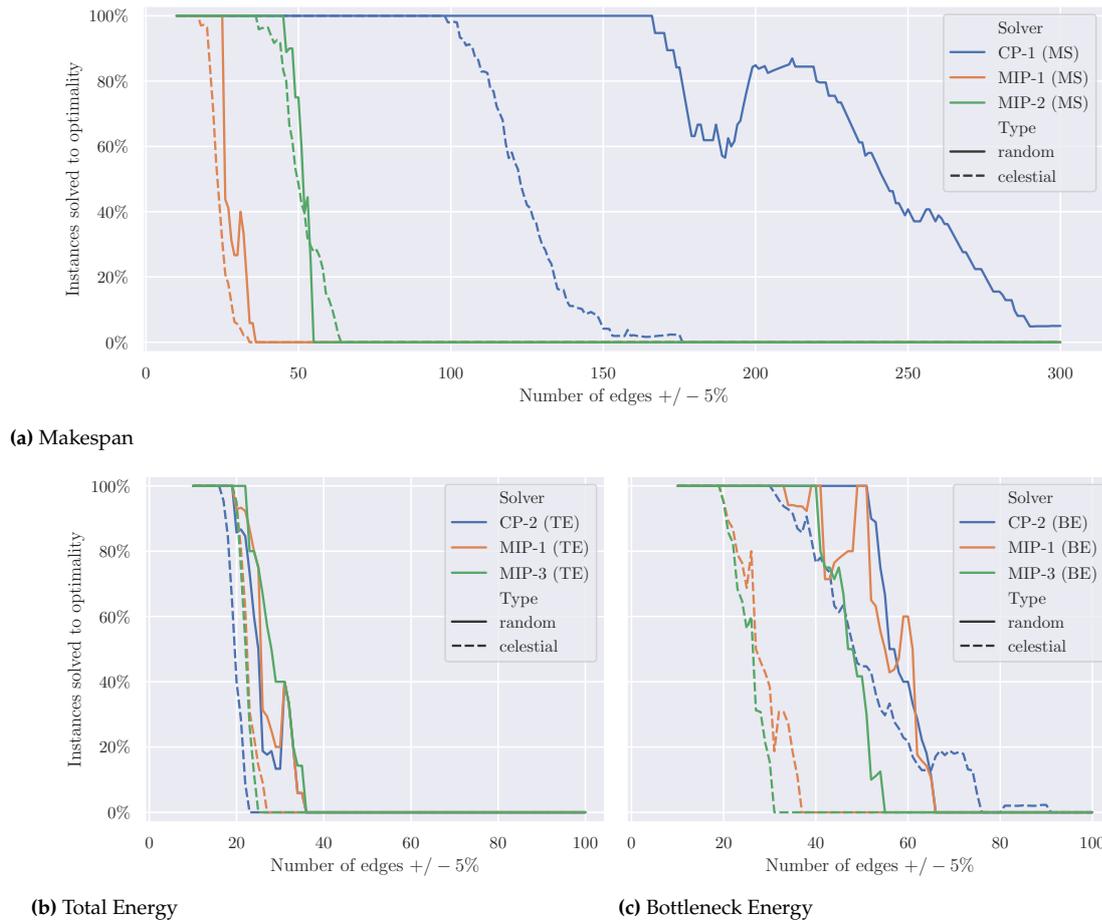


Figure 2.31: Performance of the exact solver measured in how many instances with $m \pm 5\%$ edges can be solved to provable optimality within 900 sec. The bump for CP-1 starting at 200 can be explained by the instance distribution that at this point includes more instances with lower degree.

algorithms and heuristics that provide good (but not provably optimal) solutions.

Bipartite Approximation Algorithms with Coloring Partition

The constant-factor approximation algorithms for bipartite graphs extend to general graphs by partitioning them into bipartite graphs and applying the corresponding approximation algorithm to each of the bipartite subgraphs. More specifically, assigning a vector over $\{0, 1\}$ with $\lceil \log_2 k \rceil$ bits to each color class of a k -colored graph induces a covering of its edge set with $\lceil \log_2 k \rceil$ bipartite graphs; for more details see Motwani and Naor [38]. For MSC-MS, this even preserves the approximation factor. We use the well-engineered *dsatur* heuristic [59] for the GRAPH COLORING problem, which is shipped with the *pyclustering*-package [60]. Concatenating the solutions of the bipartite graphs yields a feasible scan cover; here we use a greedy approach to minimize the transition costs. We denote this method by APX.

(Meta-)Heuristics

We also considered a number of (meta-)heuristics for optimizing the three objectives.

Greedy: Scan the first edge regarding a given or random order and then scan the edge that increases the objective the least, until all edges are scanned. If multiple edges are equally good, the first one regarding the order is selected. Many edges can be inserted without extra cost and thus the initial edge order has a strong influence on the result.

Iterated Local Search (ILS): This simple but potentially slow heuristic considers for a given start solution (in this case of *Greedy*) all possible swaps of edges; the locally best swap is carried out, until no further improvement is possible.

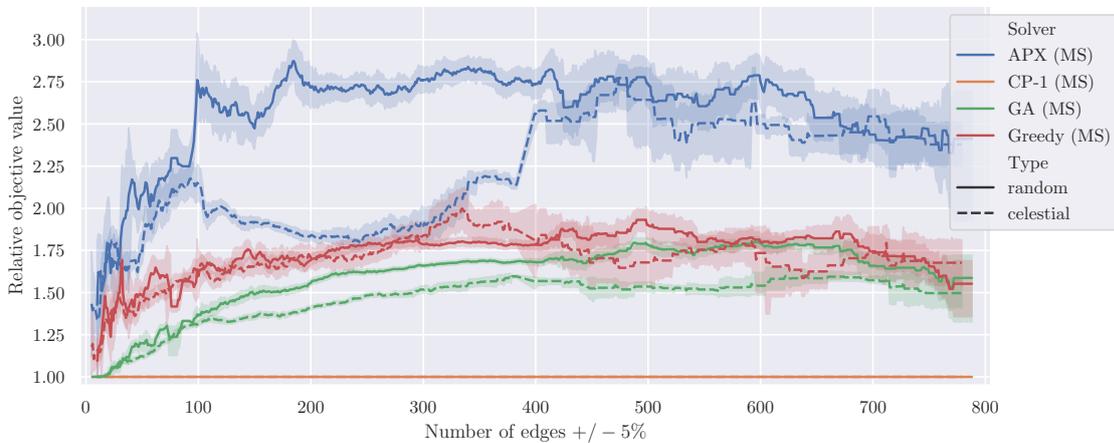
Simulated Annealing (SA): This common variation of *Iterated Local Search* performs swaps according to a probability based on the Boltzmann function $\text{Boltzmann}(T, s_1, s_2) = e^{1/T \cdot (s_2 - s_1)}$, where s_1 is the objective value of the current best solution, s_2 is the objective value of the considered solution, and $T \in \mathbb{R}^+$ is the current *temperature*. The temperature decreases over time and with it the likelihood of a worse solution being used. If the objective does not improve for some time, the temperature is increased in order to escape the local minimum. Due to randomization, we can run multiple searches in parallel. We terminate if the solution has not improved for some time.

Genetic Algorithm (GA): We start with an initial population of 200 solutions generated by a randomized Greedy. A solution is encoded by assigning each edge a fractional number between 0.0 and 1.0, similar to [61]. The scan order is determined by sorting the edges by these numbers. In each round, we build a new population by selecting the best 10% of the old population (*elitism*) and then fill the rest of the population by crossovers of the old generation. For a crossover, we select two solutions of the old generation with a probability matching their objective values (*uniform selection*) and for each edge we choose with equal probability either the number from the first or second solution (*uniform crossover*). If by chance, two edges get the same number, we randomly change one of them without influencing the order. Of the new generation of solutions, 3% are selected for mutation. A mutation applies Greedy with a probability of 60% (the old order is used as initial edge order) or changes each edge with a 3% probability to a new random number. This is repeated until we either reach a time limit of 900 sec, 300 generations, or 60 generations without improvement. The best solution found during this process is then returned.

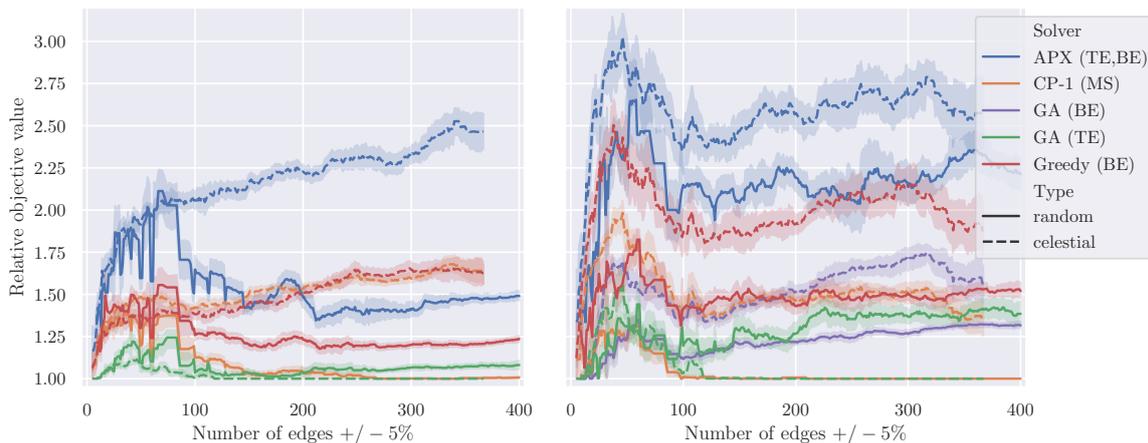
Experimental Evaluation of Approximations and Heuristics

Figure 2.32 shows experimental results for heuristically solving instances with up to 800 edges with a 900 sec time limit (at which point the current solution is returned). For MSC-MS, CP-1 yields the best results even for larger instances (where it is aborted by the time limit) by a margin of 25% to 50% to the next best algorithm, GA. For MSC-TE, the genetic algorithm

turned out to be the best approach for celestial instances by a margin of over 50 % for the larger instances. Surprisingly, CP-1 (optimizing for MSC-MS) yields slightly better solutions than the genetic algorithm for random instances of MSC-TE. The most interesting results are for MSC-BE. Here, CP-1 achieves the best results by a margin of over 20 % for random instances, and GA (TE) the best results for celestial instances by a margin of over 40 %. The excellent performance of CP-1 can be explained by a strong correlation of MSC-MS and MSC-BE for random graphs, as shown in Figure 2.33. The fact that GA (TE) is actually better in optimizing MSC-BE than GA (BE) can be explained by the weaker gradients of bottleneck objectives, because only a small part of the solution (the most expensive vertex) actually contributes to the value. However, the initial bump, at which the exact solver of MSC-BE still yields (better) solutions, indicates that these solutions could be far from optimal and that there may still be room for improvement.



(a) Makespan.



(b) Total energy.

(c) Bottleneck energy.

Figure 2.32.: Relative performance of the non-exact methods, measured by the obtained objective value divided by the best known value. We used the same instances for the exact solver, so the better denominator creates a small bump for smaller instance sizes, in particular for MSC-BE. Except for CP-1, the exact solvers did not yield good solutions for larger instances, if any at all, and are thus excluded for readability. The plots show the mean and the corresponding 95 % confidence interval. We highlight the difference between the two instance types by using different styles for the lines. Note that because these are relative values, a comparison of the performance over the different objectives is not possible. ILS and SA are excluded for readability and perform only slightly better than Greedy.

Future Work 2.4.

Can we improve the performance of the heuristics for MSC-BE by optimizing for an objective that is more sensitive to all satellites? For example, we could do a lexicographic optimization over the sorted rotation times of all satellites.

Overall, either CP-1 or GA (TE) yields the best solutions. CP-1 is especially strong on random instances for all three objectives. The approximation algorithm is usually among the worst. For MSC-MS, the algorithm performs a full rotation for nearly all instances, as $\max_{v \in V} \Lambda(v)$ is usually above 180° . Note that the factor can be worse than the approximation factor 4.5 (resp. 2), because these are not bipartite graphs.

In Figure 2.33 (first row, fourth and last column) we can additionally see that for MSC-MS the objective correlates strongly with the number of edges for celestial instances and with the average degree for random instances. Total energy seems to primarily correlate with the number of edges for both types; our random instances are on average twice as expensive. For MSC-BE, only random graphs seem to have a significant correlation to MSC-MS and the average degree.

Future Work 2.5.

Do these results carry over to three-dimensional instances or instances with heterogeneous rotation speeds?

2.7. Conclusion

In this chapter, we studied problems of MINIMUM SCAN COVER with three different and practically relevant objective functions, providing both theoretical and practical contributions: The theoretical contributions contain complexity and algorithmic results for all three objectives, showing a strong relationship to the chromatic number when minimizing the makespan. The presented practical methods are capable of computing provably optimal solutions for smaller and near-optimal solutions for larger instances.

In particular, we showed that the problems are provably hard even for simple instances, like bipartite graphs in 2D, but also provided approximation techniques whose approximation factors depend on the chromatic number in 2D and on the arboricity in higher dimensions. We developed multiple MIP and CP formulations, and demonstrated that instances of MSC-MS can be solved reliably for instances with more than 100 edges using constraint programming, which performs much better than our MIP approaches. While this approach generalizes also to 3D, we only tested 2D instances; it remains to be seen if these results are transferable to 3D. MSC-TE and MSC-BE can only be solved to optimality for much smaller instances. For solving larger instances without guarantee of optimality, we evaluated approximation algorithms and a spectrum of meta-heuristics. Within the given time limit, CP-1 provided the best solutions for all MSC-MS instances, and even the random instances for MSC-TE and MSC-BE, despite only optimizing for MSC-MS. For celestial

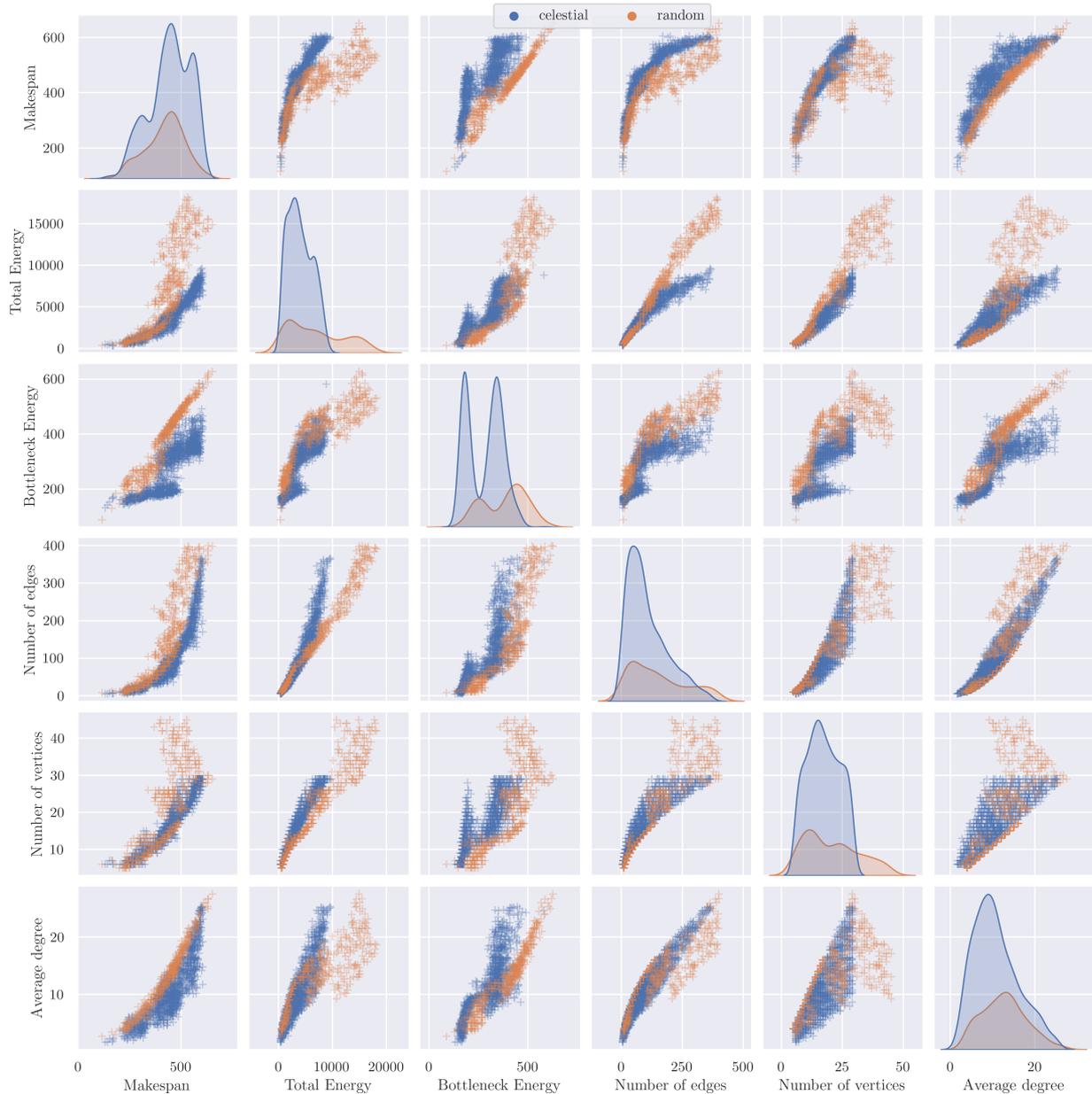


Figure 2.33.: Correlation and distribution of the best known objectives and instance properties. The diagonal shows the density distribution of the x-values. The scatter plots have a point for every existing value pair, which allows to detect correlations.

instances of MSC-TE and MSC-BE, the genetic algorithm optimizing for MSC-TE provides the best solutions. However, the results indicate perspectives for improving the optimization of MSC-BE.

At this point, fully dynamic instances (in which the vertices change their relative positions to each other over time, such as for satellites with different orbit parameters) are yet to be explored. These promise to be even more challenging, due to bigger gaps between optimal and suboptimal solutions, resulting from possibly long delays when a limited communication window has been missed.

A similar optimization problem, where we do not want to scan every edge but just broadcast a message, is considered in the next chapter.

Angular Freeze-Tag

3.

This chapter considers a problem in which a message must be broadcast with directional antennas that require expensive rotations for communication. After some basic theoretical results, we focus on solving the problem practically. Due to the superior performance of constraint programming with CP-SAT, we perform a thorough analysis of it.

3.1. Introduction

This chapter considers a problem similar to `MINIMUM SCAN COVER` that has been discussed in the previous chapter, but instead of scanning a set of edges, we want to send a message to all satellites within the network.

Providing instructions to all members of a distributed group is a fundamental task for many types of team missions. In terrestrial settings, this can usually be achieved by broadcasting to all recipients in parallel, requiring only a single transmission. However, for long-distance space missions, omnidirectional transmission can no longer be employed, due to significant loss in signal strength. Instead, transferring data is accomplished with the help of directional antennas, requiring a highly focused communication beam that is targeted directly at the intended recipient.

A striking example of such an antenna is on the space probe Voyager, as can be seen in Figure 3.1. Satellites in earth-orbit use similar, but smaller, antennas. These transmissions must be performed individually, involving maneuvers for achieving appropriate antenna orientation; the time for such a maneuver is proportional to the required angle of rotation, with negligible time for the actual transmission itself. The overall process allows one parallel component: a team member that has already been “activated” by having received the data, may relay this to other partners, motivating the use of intricate communication trees to achieve rapid dissemination of information to all members of a swarm of spacecraft.

This same method can also be utilized if we want to quickly distribute data, e.g., an important update. In the following, we consider a basic version of the problem under following assumptions: the agents are static points in the Euclidean space, there are no delays in transmission, and the transmission cone is modeled as a ray. Contrary to the model in `MINIMUM SCAN COVER` of the previous chapter, we only require the sender to point to the receiver, but the receiver does not need to be adjusted.

Problem 3.1.1 (Angular Freeze-Tag (AFT)) *Given a graph $G = (P, E)$ with $P = \{p_0, p_1, p_2, \dots\}$ being a set of agent positions in d -dimensional space, each agent $p \in P$ has an initial heading which we represent by an auxiliary point s_p that this heading aims at. At time $t = 0$, only p_0 is active, while all other agents are inactive. An agent p_i is activated by an active,*

- 3.1 Introduction 47
 - 3.1.1 Related Work 48
 - 3.1.2 Overview 49
- 3.2 Hardness 49
- 3.3 Approximation Algorithm 52
- 3.4 Exact Algorithms 52
 - 3.4.1 Mixed Integer Programming 53
 - 3.4.2 Constraint Programming . 55
 - 3.4.3 How Does CP-SAT Work? 65
- 3.5 Heuristics 68
 - 3.5.1 Random 68
 - 3.5.2 Greedy 69
- 3.6 Evaluation 72
 - 3.6.1 Benchmark 73
 - 3.6.2 Optimal Solutions 73
 - 3.6.3 Good Solutions 74
- 3.7 Adaptions for BAFT 75
 - 3.7.1 Adapting the CP 75
 - 3.7.2 Greedy Algorithm 76
- 3.8 Conclusion 77



Figure 3.1: The space probe Voyager and its directional antenna for transmitting data. (Image CC by NASA.)

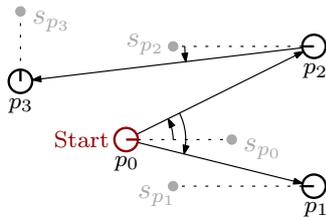


Figure 3.2.: Activating all agents by rotations in AFT: p_0 first activates p_2 which then activates p_3 , while p_0 rotates back to activate p_1 .

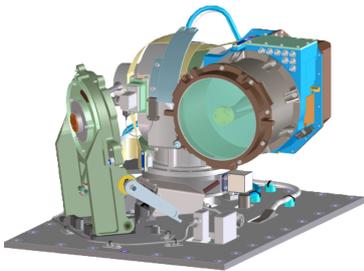


Figure 3.3.: Rendition of the optical module of the Lunar Laser Communication Demonstration that enables a satellite in lunar orbit to communicate with earth with high data rates. (Image CC by NASA.)

adjacent agent p_j , whose heading aims straight at p_i ; adjusting this heading incurs a cost equal to the required angular change. Let $\alpha(p_i p_j, p_i p_k) \in \mathbb{R}_0^+$ denote the cost of p_i to change the heading from p_j to p_k . The objective is to minimize the time T until all agents are activated, i.e., minimize the makespan of the overall activation schedule.

An example can be seen in Figure 3.2. If you desire a more thorough definition, you can also skip directly to the mixed integer programming or constraint programming formulation in Section 3.4 (Exact Algorithms) on page 52.

While it is technically possible to mix a directed antenna for sending with an omnidirectional for receiving, it is much less effective and would probably require two antennas instead of one. Also, the emerging laser communication techniques, as can be seen in Figure 3.3, appear to use the same optics for the laser and the sensor. This model is still at least theoretically feasible, and provides a nice geometric problem similar to the freeze-tag game and problem where the receiver is completely frozen until being activated. One can easily imagine a game variant of freeze-tag with water pistols that corresponds to AFT. Nonetheless, we also adapt the most promising algorithms for AFT to the (for satellites) more practical variant, in which both sides have to adjust.

Problem 3.1.2 (Bidirectional Angular Freeze-Tag (BAFT)) *Every point $p \in P$ is now allowed to rotate without activation (starting at its initial heading). A point can still only activate neighbors if it already has been activated itself. For an activation, both sides have to head toward each other at the same time.*

Future Work 3.1.

How does the problem change if the satellites can activate all agents within a cone of a specific angle? By choosing a very small angle, we can reuse the hardness results of the current variant, but for larger angles we may obtain better schedules.

3.1.1. Related Work

The ANGULAR FREEZE-TAG PROBLEM is an adaptation of the freeze-tag problem. In the freeze-tag problem, activating an inactive robot is performed by moving an active robot next to it. The objective (to minimize the makespan of the overall schedule) is the same as for our current problem, but the cost of an activation (the distance to the robot instead of the angle) is different. This problem is NP-hard even for star graphs, but there are polynomial-time approximation schemes (PTAS) for star graphs and geometrically-embedded instances [62]. Unweighted graphs are considered in [63] and a set of heuristics is evaluated in [64]. There are further results on a variant with k activated agents in [65] and on the online variant in [66].

In case this content has been skipped over, we refer the curious reader to the related work of the similar MINIMUM SCAN COVER in Subsection 2.1.2.

3.1.2. Overview

In the following, we focus on the problem in 2D and first show that AFT cannot be approximated better than $5/3$, even for bipartite graphs, and provide a 9-approximation algorithm for any graph. We also show the same hardness result for the bidirectional variant. Afterwards, we come to the primary part of this chapter, which considers computing optimal and heuristic solutions for AFT. We propose mixed integer programming and constraint programming formulations in Section 3.4. For constraint programming, we provide an explanation of the techniques used by the constraint programming solver CP-SAT, that already yielded a superior performance for MINIMUM SCAN COVER. We continue with engineering heuristics in Section 3.5, and perform an experimental evaluation of all approaches in Section 3.6. At the end of this chapter, we discuss how to adapt the constraint program and the greedy algorithm to BAFT in Section 3.7.

3.2. Approximation Hardness

We show that the AFT is computationally hard, even to approximate.

Theorem 3.2.1 *A c -approximation algorithm for the AFT with $c < 5/3$ implies $P = NP$, even for complete or bipartite graphs.*

Proof. We give a reduction from SATISFIABILITY; see Figure 3.4 for a sketch. Our construction has a solution with a makespan of 3ε if it is satisfiable and 5ε otherwise, where $\varepsilon > 0$ is a sufficiently small angle. Our construction uses five different types of agents, as follows.

- ▶ The *start agent* p_0 directly activates the *decision agents*, but does not have any other agents within 5ε .
- ▶ For each variable we have a *decision agent* and two *variable assignment agents* (one each for `true` and `false`) in opposing angles of ε , but no further agents within a 5ε rotation range. It is directly activated from p_0 .
- ▶ The *variable assignment agent* directly activates all corresponding *literal agents*, but has no further agents in a 4ε rotation range. The earliest possible activation time is ε . Only one of the two agents for each variable can be activated at time ε (by the *decision agent*), the other one has to wait an additional 2ε .
- ▶ For each literal there is a *literal agent* that has its clause agent a rotation of 2ε away, but no further agents within 4ε . The earliest possible activation time is ε .
- ▶ For each clause there is a *clause agent* that has no agent within its 2ε rotation range. Its earliest possible activation time is 3ε .

A clause agent can only be activated by its literal agents in less than 5ε , and a literal agent is either activated at ε or 3ε , depending on which of the variable assignment agents got activated first. Thus, a clause is activated at 3ε if and only if a corresponding variable agent has been activated in time; otherwise, it takes 5ε .

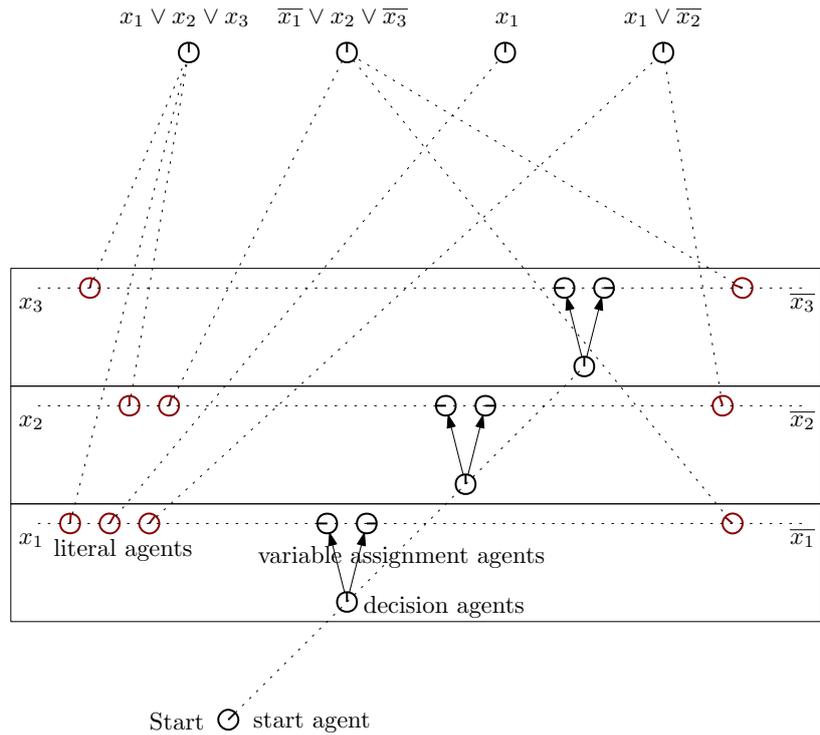


Figure 3.4.: Sketch of the hardness construction. Red variable agents are 2ϵ from their designated heading, which they can target upon activation. The decision agent for each variable is a rotation ϵ from both of its corresponding variable assignment agents. A schedule of makespan 3ϵ exists if and only if there is a satisfying truth assignment; otherwise, the makespan is at least 5ϵ .

We only need edges from the start agent to the decision agents, from the decision agents to the variable assignment agents, from the variable assignment agents to the literal agents, and from the literal agents to the clause agents. This is a bipartite graph, with the literal and the decision agents in one class, and the other agents in the other class. We can arbitrarily fill up the other edges to obtain a complete graph. \square

We can obtain the same result with a different construction for BAFT, where all satellites can rotate from the beginning, but both sides need to head toward each other for an activation.

Theorem 3.2.2 *A c -approximation algorithm for the BAFT with $c < 5/3$ implies $P = NP$, even for bipartite graphs.*

Proof. We again give a reduction from SATISFIABILITY with a corresponding sketch in Figure 3.5. Five kinds of agents are placed at (nearly) the same positions: The *start agent*, one *variable agent* for each variable, one *positive literal agent* for every positive literal occurrence in each clause, analogous *negative literal agents* for negative literals, and one *clause agent* for each clause. If a positive literal occurs in, e.g., 3 clauses, then there are three corresponding positive literal agents. The positions of the variable agents, positive literal agents, negative literal agents, and clause agents form a square with 90° angles, with the start agent placed in the middle. The start agent is directed at and connected with all of the variable agents. The variable agents are directed at the start, and are additionally connected to their corresponding positive and negative literal agents. The positive and negative literal agents are heading toward their corresponding variable agent, and are additionally only connected to their one clause at an angle of 90° . Finally, the clause agents head away from the construct and are only connected to their positive and negative literal agents. They need

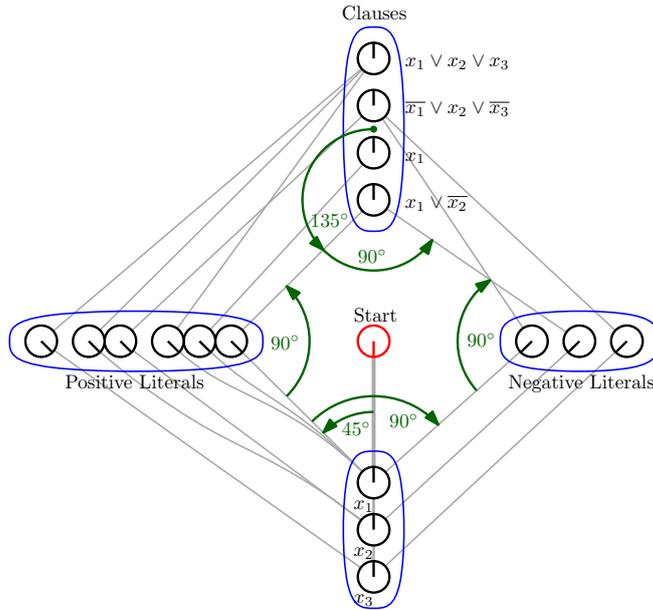


Figure 3.5: Hardness construction for BAFT. Edges are shown in gray. The agents within the blue circles are nearly at the same position. The whole construction forms a square with the start agent in the middle. See text for construction description.

135° to reach either the positive or negative literals, and the positive and negative literals are 90° apart from each other.

If the formula is satisfiable, we can create a schedule of 135° rotation time as follows: The variable agents are activated immediately and rotate toward their positive or negative literal agents, according to their assignment. The corresponding satisfied literal agents are activated after 45° rotation time. Their only non-activated neighbor is the clause it is assigned to, which it can activate after a 90° rotation after 135° rotation time. Because we assumed the assignment to be satisfying, every clause agent is activated at that time. In parallel, the variable agents rotate to the opposite literal agents and also activate them after a 90° rotation after 135° rotation time. Thus, all agents are activated after 135° rotation time, and this is the fastest way possible.

To conclude the proof, we have to show that a schedule of 135° rotation time implies the existence of a feasible assignment and that the next best schedule has a rotation time of 225° .

The literal agents are only connected to their variable agent and to their assigned clause agent. Thus, they can only activate their clause agent, and this only 90° after their own activation. They may be activated by their clause agent, but can activate no other agents in this case. This implies that every literal agent is either activated at 45° or 135° . If all literal agents of a clause are activated after 135° rotation time, the clause agent can only be activated after 225° rotation time. Otherwise, the clause agent can be activated after 135° rotation time, but not earlier. By this construction, only the positive or the negative literal agents can be activated after 45° rotation time.

If we have a schedule of 135° rotation time, every clause agent needs to have a literal agent that has been activated within 45° rotation time. This can happen only by activation by the corresponding variable agent, which can only activate its positive or the negative literals within the time, but not both. A literal agent always needs a 90° rotation to activate its clause, because if it is activated by its only other neighbor, the variable agent, it

needs 90° to rotate. If it is activated by its clause agent, it has no further neighbors to activate. This implies a satisfiable variable assignment. Due to the square construction, all activations happen at $45^\circ + i \cdot 90^\circ$ (assuming no unnecessary pauses are made), and if an activation of all clause literals has not been possible until 135° , the makespan is at least 225° . \square

3.3. Approximation Algorithm

We can provide a simple constant factor approximation, based on a result by Beck [67] on the *linear search problem*. In that scenario, an agent has to locate a hidden object in a one-dimensional environment; from a given starting location, the best strategy for this online problem is to alternate between going left and right, while doubling the search depth in each iteration. This yields a total search distance that is within a factor of 9 of the optimum.

Theorem 3.3.1 *There is a 9-approximation algorithm for the AFT in 2D, even for unknown agent locations and headings, assuming a lower bound of $\varepsilon > 0$ for the rotational angle of any activating agent.*

Proof. As soon as an agent is activated, it follows the doubling strategy (starting at ε) from linear search, carried out for rotation. Thus, it follows by induction, that any agent p_i that gets activated by T_i in an optimal schedule is activated within $9T_i$. This is due to the fact that every predecessor in the optimal solution needs at most nine times the necessary rotations in the approximated solution. Additionally, it does not matter if another agent activates the agent before the optimal predecessor, as the receiver does not have to adjust in the variant of this problem. This approach is oblivious to the actual underlying graph, and works for every connected graph. \square

Note that we cannot apply the refined technique by Bose et al. [68] for linear search, as it requires both an upper and a lower bound on the search distance.

Future Work 3.2.

Can we obtain an approximation algorithm for the bidirectional variant? Can we obtain an approximation algorithm for three-dimensional instances?

3.4. Exact Algorithms

This problem can be expressed as mixed integer program (MIP) and as constraint program (CP), which allows us to obtain optimal solutions for many instances using advanced solvers, such as Gurobi [69] or Google's OR-Tools [70]. For many hard problems, such techniques can still yield optimal solutions for reasonably large instances. Even if no optimal solution can be achieved, often at least a good solution including bounds can be computed. If sufficiently large instances can be solved

with these techniques, we can consider ourselves lucky and do not need to investigate further algorithms. If only smaller instances can be solved, these solutions can still be valuable for theoretical analysis and further algorithm development. In this chapter, we also take a deeper look into the performance of constraint programming for such problems after the superior performance for MINIMUM SCAN COVER in the previous chapter.

While MIPs and CPs look very similar in their formulation, their underlying techniques, and therefore their performance, can be fundamentally different. Mixed integer programming is based on linear programming, and can utilize many powerful theoretical insights. MIPs are usually solved by branch and cut algorithms based on the linear relaxation. Constraint programming formulations are a superset of mixed integer programming formulations, and our solver, CP-SAT of Google OR-Tools, utilizes a SAT-solver in the background to solve them. While every MIP can be expressed as a CP, most constraint programming solvers allow more advanced constraints. The more specific the constraint is, the better the chances are that the solver can utilize powerful propagators. At the same time, CP-SAT does not allow callbacks such that common exponential constraints for MIPs, e.g., the subtour constraints of Dantzig [58], cannot be efficiently implemented. We now first present the MIP-formulation and then a CP-formulation, for which we do some additional evaluations to make the best design decisions. The performance of both approaches is compared in Subsection 3.6.2.

3.4.1. Mixed Integer Programming

In this section, we describe how to formulate the ANGULAR FREEZE-TAG PROBLEM as a mixed integer program to be solved with, e.g., Gurobi or CPLEX.

First, we need fractional variables that represent the activation times of each agent, see Figure 3.6. Let these variables be denoted by $t_p \in \mathbb{R}_0^+$ for $p \in P$. The activation time variable for the start agent p_0 can directly be replaced by the constant 0.0, as it is active from the beginning.

This already enables us to formulate the objective function:

$$\min \max_{p \in P} t_p \quad (3.1)$$

A min max-objective can be implemented as a linear program by minimizing an auxiliary variable $m \in \mathbb{R}_0^+$ and adding the constraints $m \geq t_p$ for all activation time variables t_p with $p \in P$.

Next, we need to encode the time dependencies. Let p activate its neighbor n . If n is the first neighbor that p activates, the activation time difference $|t_p - t_n| = t_n - t_p$ needs to be at least the time that p needs to rotate to n from its start position. If p activates n' before it activates n , the activation time difference $|t_n - t_{n'}| = t_{n'} - t_n$ needs to be at least the time that p needs to rotate from n' to n .

To enforce these constraints, we first need to encode the rotation pattern of each agent $p \in P$. To simplify the notation, let $H(p) = N(p) \cup \{s_p\}$ be the extended neighborhood including a virtual neighbor for the initial heading. Let $x_{n,n'}^p \in \mathbb{B}$ denote if p activates its neighbor $n' \in N(p)$ after

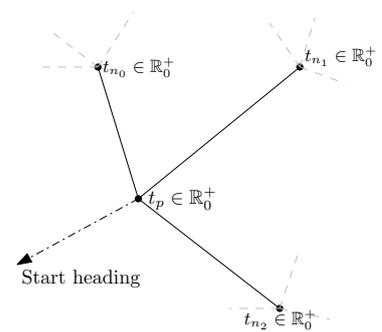


Figure 3.6.: Every agent gets an activation time variable.

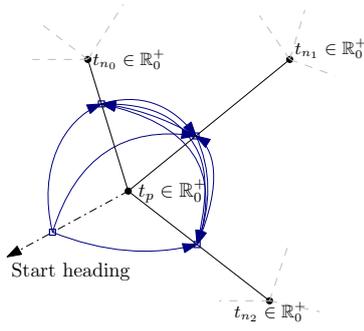


Figure 3.7.: The rotation pattern only needs direct rotations (blue) between neighbors and start headings.

heading to $n \in H(p)$. This gives us a small set of directed rotations that are sufficient to encode any optimal rotation pattern, see Figure 3.7.

Enforcing an agent $p \in P - p_0$ to get activated is simply enforcing that one of its neighbors rotates toward it.

$$\forall p \in P - p_0 : \sum_{p' \in N(p)} \sum_{h \in H(p')} x_{h,p}^{p'} = 1 \quad (3.2)$$

Of course, p_0 is already activated from the beginning and does not need to be activated again. This can still allow activation cycles, where a cycle of non-activated agents activate each other. To prevent this, we can add the constraints

$$\forall S \subseteq P, p_0 \in S : \sum_{p \in S} \sum_{\substack{n \in H(p) \\ n' \in N(p) \setminus S}} x_{n,n'}^p \geq 1 \quad (3.3)$$

which enforce that for every subset of agents $S \subseteq P$ that contains the start agent p_0 but is incomplete, i.e., $S \neq P$, an activation of the remaining agents $P \setminus S$ by S must happen. Every activation chain, hence, is rooted in p_0 .

A further problem that can occur is that the rotation pattern of an agent is infeasible. Every rotation pattern for an agent $p \in P$ needs to begin at the start heading and be coherent, i.e., be a directed Hamiltonian path on a subset of agents of $H(p)$ originating at s_p . Such paths are a common element of many combinatorial problems and their corresponding integer programming formulations. The corresponding constraints are, hence, relatively straight forward:

If p rotates from $n \in N(p)$ to $n' \in N(p)$,

1. n cannot have further successors and
2. n needs a predecessor $n'' \in H(p)$.

This can be expressed by

$$\forall p \in P, n \in N(p) : \sum_{n'' \in H(p)} x_{n'',n}^p \geq \sum_{n' \in N(p)} x_{n,n'}^p \quad (3.4)$$

and

$$\forall p \in P, n \in N(p) : \sum_{n'' \in H(p)} x_{n'',n}^p \leq 1 \quad (3.5)$$

We also have to make sure that Equation 3.4 is not tricked by cycles, but the rotation pattern is indeed a path starting at s_p . This can be performed by classical subtour elimination constraints.

$$\forall p \in P, S \subseteq N(p) : \sum_{n,n' \in S} x_{n,n'}^p \leq |S| - 1 \quad (3.6)$$

We now have feasible rotation patterns at each agent, which already describe a feasible but arbitrary solution. For optimizing the makespan, we need to synchronize the activation time variables with the rotation patterns: whenever we perform a rotation, the necessary rotation times must propagate to the activation time variables, see Figure 3.8.

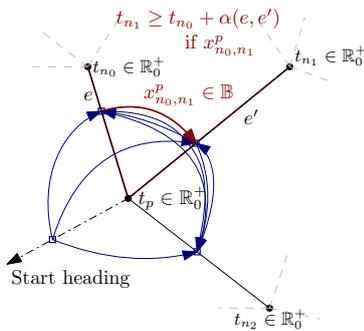


Figure 3.8.: Whenever we do a rotation, the necessary rotation times need to propagate to the activation times.

If p activates n first, i.e., $x_{s_p,n}^p = \text{true}$, we need to enforce that t_n is at least t_p plus the time p needs to rotate to n from its initial heading.

$$\forall p \in P, n \in N(p) : t_n \geq t_p + \alpha(p s_p, pn) \quad \text{if } x_{s_p,n}^p \quad (3.7)$$

If p activates n after activating n' , i.e., $x_{n',n}^p = \text{true}$, we need to enforce that t_n is at least $t_{n'}$ plus the time p needs to rotate from n' to n .

$$\forall p \in P, n \neq n' \in N(p) : t_n \geq t_{n'} + \alpha(p n', pn) \quad \text{if } x_{n',n}^p \quad (3.8)$$

Of course, we cannot implement conditional constraints directly, but we can make use of the Big-M method. Let T_{UB} be an upper bound on the optimal makespan, e.g., computed by a greedy algorithm. Whenever, the corresponding rotation variable is false, the inequality are trivially fulfilled by subtracting the upper bound. This results in the following linear constraints:

$$\forall p \in P, n \in N(p) : t_n \geq t_p + \alpha(p s_p, pn) - (1 - x_{s_p,n}^p) \cdot T_{UB} \quad (3.9)$$

$$\forall p \in P, n \neq n' \in N(p) : t_n \geq t_{n'} + \alpha(p n', pn) - (1 - x_{n',n}^p) \cdot T_{UB} \quad (3.10)$$

Combining all of these constraints results in a feasible and complete (regarding optimality) description of the solution space as a mixed integer program that can be solved by corresponding solvers.

3.4.2. Constraint Programming

We now describe how one can formulate the problem as a constraint program that can be solved using CP-SAT¹. After the formulation, we directly analyze some design decisions via some quick experimental evaluations.

There are two important pieces of information we need for a solution: (1) when does an agent get activated, and (2) by which agent does it get activated. For this, we use two types of variables.

1. $t_p \in \mathbb{R}_0^+$ for $p \in P$ denotes the activation time. An upper bound for these variables is computed by the greedy algorithm (see Subsection 3.5.2). We directly replace p_0 with the constant 0, because it is activated in the beginning.
2. $x_{p \rightarrow p'} \in \mathbb{B}$ for $p \in P, p' \in N(p) - p_0$ states if p activates p' . We exclude $p' = p_0$ as p_0 does not need activation. Contrary to the MIP, we do not explicitly express the concrete rotations of the satellites. The rotations are implicitly contained in the activation time variables.

The objective function can be expressed as

$$\min \max_{p \in P} t_p. \quad (3.11)$$

We discuss the two options to implement such an objective in CP-SAT in a later part of this section. Let us now add the corresponding constraints.

1: Note that, contrary to MIPs, the set of allowed constraints is not strictly defined for CPs, and depends on the concrete solver. The following section assumes that we use CP-SAT.

First, we ensure that every agent gets activated.

$$\forall p \in P - p_0 : \sum_{n \in N(p)} x_{n \rightarrow p} = 1 \quad (3.12)$$

To make sure that the x -variables actually are a tree and do not contain cyclic activations, we add an additional class of variables. Let $i_p \in \mathbb{N}^{|V|}$ denote the order in the tree with $i_{p_0} = 0$, such that children have higher numbers than their parents. We can enforce correct values by

$$\forall p \in P, p' \in N(p) - p_0 : x_{p \rightarrow p'} \Rightarrow i_p \leq i_{p'} - 1. \quad (3.13)$$

Now we only have to ensure that the scan times are actually feasible. The following constraint ensures that the activating agent has enough time to rotate from s_p .

$$\forall p \in P, p' \in N(p) - p_0 : x_{p \rightarrow p'} \Rightarrow t_{p'} \geq t_p + \alpha(p s_p, p p') \quad (3.14)$$

Due to the triangle inequality, it does not matter if p' is actually the first to be activated by p . The next constraint ensures that between any two activations performed by an agent, there is sufficient time to rotate. Note that this constraint also requires α to fulfill the triangle inequality, as it is indifferent regarding the concrete order.

$$\begin{aligned} \forall p \in P, n, n' \in N(p) - p_0, n \neq n' : \\ x_{p \rightarrow n} \wedge x_{p \rightarrow n'} \Rightarrow |t_n - t_{n'}| \geq \alpha(p n, p n') \end{aligned} \quad (3.15)$$

CP-SAT actually does not support composite expressions for if-conditions and absolute values, such that we need to introduce auxiliary variables for these.

We now continue to analyze the accuracy, influence of the resolution, importance of tight variable bounds, usage of more powerful constraints, modelling of the min max-objective, and the determinism of CP-SAT. This part purposefully goes deeper on the techniques of CP-SAT than the other chapters to highlight not only the usage of constraint programming for AFT, but also more general properties of CP-SAT that can carry over to other optimization problems. Readers not interested in the details of constraint programming can skip to Section 3.5 (Heuristics) on page 68.

Accuracy of CP-SAT

A restriction of CP-SAT is that it only allows integral variables and constants. Thus, we need to convert the fractional values to large integral numbers via multiplying and rounding by a large number. Reasonably large numbers result in an accuracy that is comparable to Gurobi, which operates on inexact floating point numbers. Gurobi performs some automatic optimizations to improve the numeric accuracy and performance of the given input. For CP-SAT, we have to do these optimizations by ourselves. In the remainder of the section, we analyze the accuracy and performance of CP-SAT for different resolutions.

Continuous equality constraints can become problematic with CP-SAT. They induce an intersection of two hyperplanes, which is most likely not perfectly aligned with the integral grid. In the worst case scenario, this can lead to infeasibility in the integral domain. MIP-solvers, like Gurobi,

automatically introduce feasibility gaps. For CP-SAT, we potentially have to do this ourselves. Fortunately, this problem does not apply to AFT. The only part we need to worry about is the solution quality, because the inequality constraints can potentially propagate and accumulate rounding errors: activation times depend on other activation times, which themselves depend on other activation times, and every such step can introduce a small error. We now prove that this error remains reasonably small.

We know that every value can be off by at most one unit in the integral domain (by rounding either up or down). This allows us to limit the maximal absolute error linear to the number of agents. Let $I(x)$ be a rounding function that maps x either to $\lceil r \cdot x \rceil$ or $\lfloor r \cdot x \rfloor$.

Lemma 3.4.1 *If the CP-formulation is integralized by I with resolution r , then a solution with makespan T in the original formulation can be translated into a solution in the integral domain with a makespan of at most $r \cdot T + 3 \cdot |P| - 3$.*

Proof. Consider a solution in the original formulation with makespan T , and let $t_{p_1} \leq t_{p_2} \leq \dots \leq t_{p_{|P|-1}}$ be sorted. We show that $I'(t_{p_i}) = \lfloor r \cdot t_{p_i} \rfloor + 3 \cdot i$ integralizes the time variables, such that all constraints after integralization are still satisfied. The makespan of the corresponding solution is by definition $\lfloor r \cdot t_{p_{|P|-1}} \rfloor + 3 \cdot (|P| - 1)$, and thus at most $r \cdot T + 3 \cdot |P| - 3$. The only influenced constraints are in Equation 3.14 and Equation 3.15. For these, we need to show that the equations are still satisfied after integralization.

For Equation 3.14, we need to show that

$$t_{p_j} \geq t_{p_i} + \alpha(p_i s_{p_i}, p_i p_j) \Rightarrow I'(t_{p_j}) \geq I'(t_{p_i}) + I(\alpha(p_i s_{p_i}, p_i p_j)) \quad (3.16)$$

which can be seen as follows:

$$\begin{aligned} I'(t_{p_j}) &= \lfloor r \cdot t_{p_j} \rfloor + 3j \\ &\geq \lfloor r \cdot (t_{p_i} + \alpha(p_i s_{p_i}, p_i p_j)) \rfloor + 3j \\ &\geq \lfloor r \cdot t_{p_i} \rfloor + \lfloor r \cdot \alpha(p_i s_{p_i}, p_i p_j) \rfloor + 3j \\ &\geq \lfloor r \cdot t_{p_i} \rfloor + I(\alpha(p_i s_{p_i}, p_i p_j)) - 1 + 3j \\ &\geq I'(t_{p_i}) - 3i + I(\alpha(p_i s_{p_i}, p_i p_j)) - 1 + 3j \\ &\geq I'(t_{p_i}) + I(\alpha(p_i s_{p_i}, p_i p_j)) - 1 + 3(j - i) \\ &\geq I'(t_{p_i}) + I(\alpha(p_i s_{p_i}, p_i p_j)) - 1 + 3 \\ &\geq I'(t_{p_i}) + I(\alpha(p_i s_{p_i}, p_i p_j)) \end{aligned}$$

For Equation 3.15, we need to show that

$$\begin{aligned} |t_{p_k} - t_{p_j}| &\geq \alpha(p_i s_{p_i}, p_i p_j) & (3.17) \\ \Rightarrow \left| I'(t_{p_k}) - I'(t_{p_j}) \right| &\geq I(\alpha(p_i s_{p_i}, p_i p_j)) \end{aligned}$$

Note that for $k > j$ and therefore $t_{p_k} \geq t_{p_j}$ and $I'(t_{p_k}) \geq I'(t_{p_j})$, we can rearrange the inequalities to Equation 3.16 (by exchanging the rotation

cost). Thus, we only need to prove the case that $j < k$.

$$\begin{aligned}
\left| I'(t_{p_k}) - I'(t_{p_j}) \right| &= I'(t_{p_k}) - I'(t_{p_j}) \\
&= (\lfloor r \cdot t_{p_k} \rfloor + 3k) - (\lfloor r \cdot t_{p_j} \rfloor + 3j) \\
&= (\lfloor r \cdot t_{p_k} \rfloor - \lfloor r \cdot t_{p_j} \rfloor) + 3(k - j) \\
&\geq (r \cdot (t_{p_k} - t_{p_j}) - 2) + 3(k - j) \\
&\geq (r \cdot (t_{p_k} - t_{p_j}) - 2) + 3 \\
&\geq r \cdot (t_{p_k} - t_{p_j}) + 1 \geq r \cdot \alpha(p_i s_{p_i}, p_i p_j) + 1 \\
&\geq I(\alpha(p_i s_{p_i}, p_i p_j)) - 1 + 1 \geq I(\alpha(p_i s_{p_i}, p_i p_j))
\end{aligned}$$

□

Lemma 3.4.2 *A solution with makespan T_r in the CP-formulation integralized by I with a resolution r can be translated into a feasible solution of the original formulation with makespan $\frac{T_r}{r} + \frac{|P|-1}{r}$.*

Proof. Consider a solution in the integralized formulation with makespan T_r , and let $t_{p_1} \leq t_{p_2} \leq \dots \leq t_{p_{|P|-1}}$ be sorted. $I'^{-1}(t_{p_i}) = \frac{t_{p_i} + i}{r}$ deintegralizes the time variables such that all constraints in the original formulation are satisfied, and the makespan is (by definition) $\frac{t_{p_{|P|-1}} + (|P|-1)}{r} \leq \frac{T_r}{r} + \frac{|P|-1}{r}$. Again, the only critical constraints are in Equation 3.14 and Equation 3.15.

Let p_i activate p_j , then the integralized solution fulfills $t_{p_j} \geq t_{p_i} + I(\alpha(p_i s_{p_i}, p_i p_j))$ and $i < j$.

$$\begin{aligned}
I'^{-1}(t_{p_j}) &\geq \frac{1}{r} \cdot t_{p_j} + \frac{j}{r} \geq \frac{1}{r} \cdot (t_{p_i} + I(\alpha(p_i s_{p_i}, p_i p_j))) + \frac{j}{r} \\
&\geq \frac{1}{r} \cdot (t_{p_i} + r \cdot \alpha(p_i s_{p_i}, p_i p_j) - 1) + \frac{j}{r} \\
&\geq \frac{1}{r} \cdot t_{p_i} + \alpha(p_i s_{p_i}, p_i p_j) - \frac{1}{r} + \frac{j}{r} \\
&\geq I'^{-1}(t_{p_i}) - \frac{i}{r} + \alpha(p_i s_{p_i}, p_i p_j) - \frac{1}{r} + \frac{j}{r} \\
&\geq I'^{-1}(t_{p_i}) + \alpha(p_i s_{p_i}, p_i p_j) - \frac{1}{r} + \frac{j-i}{r} \\
&\geq I'^{-1}(t_{p_i}) + \alpha(p_i s_{p_i}, p_i p_j)
\end{aligned}$$

Thus, the constraints in Equation 3.14 are fulfilled after deintegralization.

If p_i and p_j are activated by p_k , we have $|t_{p_j} - t_{p_i}| \geq I(\alpha(p_k p_i, p_k p_j))$. In the case that $t_{p_j} \geq t_{p_i}$, we can rearrange the equation to $t_{p_j} \geq t_{p_i} + I(\alpha(p_k p_i, p_k p_j))$, which equals the previous case but with a different angle. We therefore only need to consider the case that $t_{p_j} < t_{p_i}$ and,

hence, $i > j$.

$$\begin{aligned}
 \left| I'^{-1}(t_{p_j}) - I'^{-1}(t_{p_i}) \right| &\geq I'^{-1}(t_{p_i}) - I'^{-1}(t_{p_j}) \\
 &\geq \left(\frac{t_{p_i}}{r} + \frac{i}{r} \right) - \left(\frac{t_{p_j}}{r} + \frac{j}{r} \right) \\
 &\geq \frac{t_{p_i} - t_{p_j}}{r} + \frac{i - j}{r} \\
 &\geq \frac{I(\alpha(p_k p_i, p_k p_j))}{r} + \frac{i - j}{r} \\
 &\geq \frac{\lfloor r \cdot \alpha(p_k p_i, p_k p_j) \rfloor}{r} + \frac{i - j}{r} \\
 &\geq \frac{r \cdot \alpha(p_k p_i, p_k p_j) - 1}{r} + \frac{i - j}{r} \\
 &\geq \alpha(p_k p_i, p_k p_j) - \frac{1}{r} + \frac{i - j}{r} \\
 &\geq \alpha(p_k p_i, p_k p_j)
 \end{aligned}$$

□

Together, these two lemmas allow us to bound the maximal error.

Theorem 3.4.3 *The constraint program after integralization by I , with resolution r , returns a solution with a maximal absolute error of $\frac{4 \cdot |P| - 4}{r}$.*

Proof. Let the optimal solution have a makespan of T and the optimal solution in the integralized formulation a makespan of T_r . By Lemma 3.4.2, this yields a solution with a makespan T' with $T' \leq \frac{T_r}{r} + \frac{|P|-1}{r}$. At the same time, the solution found by the integralized formulation must have an integralized makespan of $T_r \leq r \cdot T + 3 \cdot |P| - 3$ by Lemma 3.4.1. Thus, $T' \leq \frac{r \cdot T + 3 \cdot |P| - 3}{r} + \frac{|P|-1}{r} = T + \frac{4 \cdot |P| - 4}{r}$. □

Theorem 3.4.3 allows us to use any rounding function, including $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$. But which one is the best? To evaluate this, we compare the accuracy of the three classical rounding functions on a low resolution of $r = 100$, with solutions computed on a high resolution of $r = 1\,000\,000$. The actual solution values are computed from the activation order, and not just rounding back the objective returned by CP-SAT, which would have a higher error.

Figure 3.9 shows that rounding down (floor) has a slightly higher error than the other functions, but despite using a very low resolution, the solutions are usually well below 0.5% off. The error first grows slightly for larger instances but then seems to stall. This can be explained by the fact that these are relative errors, and the average objectives for sparse larger instances increase. The runtime (not plotted) of all options is nearly identical, with rounding down being slightly faster.

However, all options have a negligible error for higher resolutions, and rounding down has two important advantages:

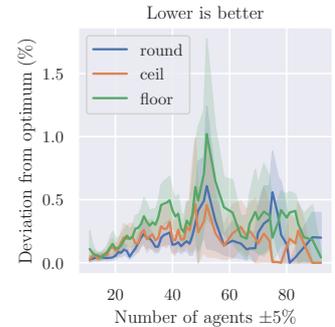


Figure 3.9: The relative error of solutions computed on low resolution ($r = 100$) for different rounding functions (rounding to closest, rounding up, rounding down).

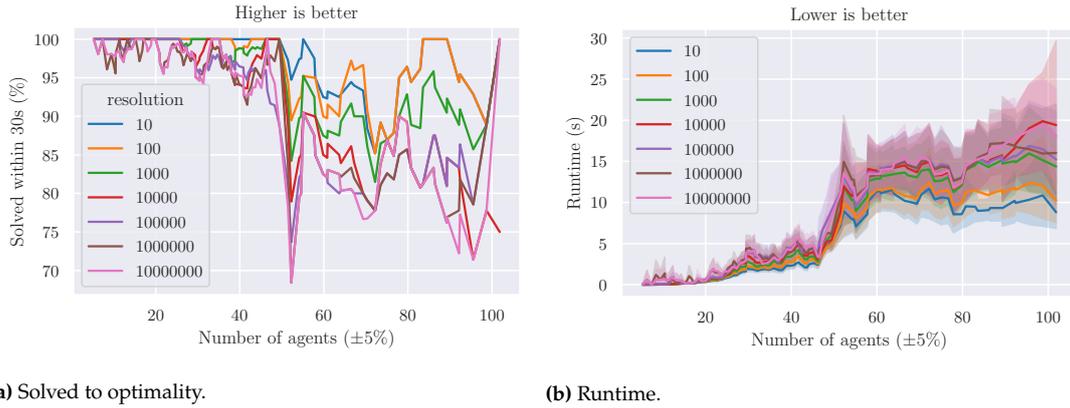


Figure 3.10.: Comparing the runtime performance of CP-SAT for different resolutions. An increased resolution influences the runtime by at most a logarithmic factor. For larger resolutions, the runtimes are very similar.

1. The lower bound returned by CP-SAT can simply be divided by r , and remains provably correct because all necessary rotations in it have just been rounded down. For the other rounding options, we have to incorporate the possible error.
2. Any upper bound remains feasible by simply multiplying it with the resolution. Variables have to be equipped with an upper bound, and close upper bounds improve the performance, as we see soon. For the other rounding options, we have to add an additional gap to the upper bound, see Lemma 3.4.1.

In the following, we settle with rounding down for integralization, i.e., $I(x) = \lfloor r \cdot x \rfloor$. How much influence does the resolution have on the solution quality and runtime? Do we need a very high resolution or does a lower resolution suffice while potentially being faster? To answer these questions, we executed the CP for the resolutions $r = 10, 100, 1000, 10\,000, \dots, 10\,000\,000$.

Let us first consider the runtime. In Figure 3.10, we see that the resolution has a surprisingly little influence on the runtime. While the very low resolutions of 1000 or less have a runtime advantage, the larger resolutions are fairly close, such that at most a logarithmic dependency could be deduced. This observation is surprising when considering that CP-SAT uses a SAT-solver in the background and needs to convert numeric variables into a set of boolean variables. CP-SAT uses *lazy clause generation*, in which CP-SAT lazily adds for a variable x with potential value v the boolean variables representing $x = v$ and $x \leq v$ (more on this later). However, all relevant values for the variables in our formulation are actually a sum of rotation angles, e.g., $t_p = \alpha(e_0, e_1) + \alpha(e_1, e_2) + \alpha(e_2, e_3) \dots$, which in combination with proper bounding and pruning can drastically reduce the number of necessary variables. Note that this is also true for many other combinatorial problems, and that the likelihood of different values being rounded to the same number decreases quickly with a higher resolution. This implies that one should directly solve the optimization problem using the desired or higher resolution and not try shortcuts like dynamically increasing the resolution in multiple iterations.

In Figure 3.11, we can see that we already obtain the optimal solutions

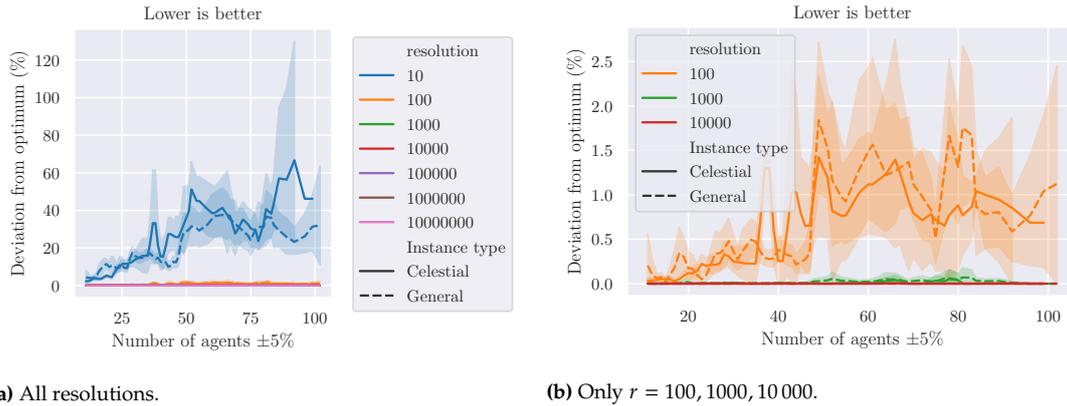


Figure 3.11.: Deviation from the optimum for different resolutions and instance types. For resolutions of $r = 10\,000$ and above, we always obtained the optimal solutions.

for reasonably low resolutions. Only for $r = 10$ are there significant deviations. While there are still some deviations for $r = 1000$, all higher resolutions only returned the optimal solutions. This also indicates that the optimal solutions for the considered instances are reasonably segregated from non-optimal instances. Hence, solving instances with reasonably low resolutions is a valid option, but the runtime advantage is low. Because we know that the potential error is linear, another option would be to dynamically scale the resolution based on instance size. We choose a constant resolution of $r = 1\,000\,000$ for the remaining experiments.

Performance-Influence of Upper Bound Tightness

Integral variables in CP-SAT have a lower and upper bound. The lower bound for this problem (trivially) is zero. An upper bound can be obtained by a greedy algorithm because no activation time is larger than the makespan. A question that directly comes to mind is how much energy should we put into obtaining good upper bounds? Is it worthwhile to spend extra energy on a tight upper bound?

To answer these questions, we perform the following experiment: We compute a greedy solution (see Section 3.5.2 (Greedy) on page 69) as an initial upper bound and use it as baseline. We then solve the constraint program with increased upper bounds, starting small with 5 and 10%. Afterwards, we double it multiple times and do a final run with 512 times the greedy upper bound.

The results with a time limit of 10s can be seen in Figure 3.13. The plots show that the tight upper bounds of 1.0, 1.05, and 1.1 times the greedy solution have a significant advantage. Already for the doubled upper bound, the runtime nearly doubles (note that the runtime plot is biased as it also contains timeouts). For even larger upper bounds, the differences get smaller, but for tight bounds already 5% results in a visible decrease in performance. This implies that a tight upper bound has a significant influence on the performance of the CP, and getting it closer by a few percent can already pay off.

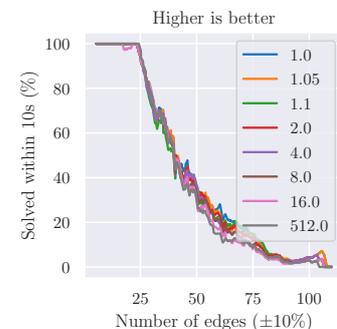


Figure 3.12.: Instances solved to optimality using the MIP in percent for different upper bounds. The influence seems to be much smaller than for CP-SAT.

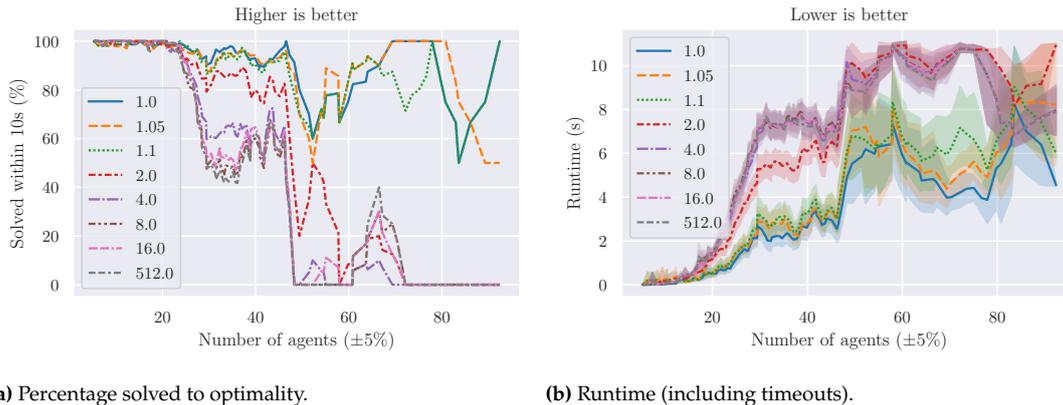


Figure 3.13.: Performance of using multiples (1.0, 1.05, 1.1, 2.0, ...) of the upper bounds for the variable ranges in the constraint program. We can see a strong decrease in performance with weaker upper bounds.

For the Big-M formulation of MIPs it is known that a smaller M (based on the upper bound) can improve the runtime. However, the influence on the MIP seems to be much lower than for CP-SAT, as can be seen in Figure 3.12. Of course, the MIP seems to perform much worse in general for this problem, so the influence of the upper bound can be negligible because it struggles in other parts.

Providing CP-SAT with the initial solution as variable hints showed a negative effect in our first experiments. The tight upper bound suffices to speed up the optimization process, while (non-optimal) variable hints seem to lead CP-SAT's search process astray.

Benefit of Using AllDifferent

The $i_p, p \in P$ variables order the agents within the tree such that parents have a lower index than the children, enforced by the constraints in Equation 3.13. A potential idea is to directly enforce that the vertices have a strict order with unique indices, i.e., we have a unique list of agents that corresponds to the activation order. This can be easily implemented by using the AllDifferent-constraint of CP-SAT.

2: <https://developers.google.com/optimization/cp/cryptarithmic> and <https://developers.google.com/optimization/cp/queens>

$$\text{AllDifferent}(i_{p_0}, i_{p_1}, \dots)$$

The AllDifferent-constraint is useful for efficiently expressing many problems and is actually used in both major examples of CP-SAT². Thus, it is reasonable to assume that it is highly optimized and can speed up satisfying the constraints in Equation 3.13. To follow this lead, we compare the runtime of the original formulation with the runtime of the formulation with the added AllDifferent-constraint. The results in Figure 3.14 show that AllDifferent seems to slow down CP-SAT instead of speeding it up. The idea of speeding up the CP with the expressive AllDifferent-constraint can, therefore, be discarded.

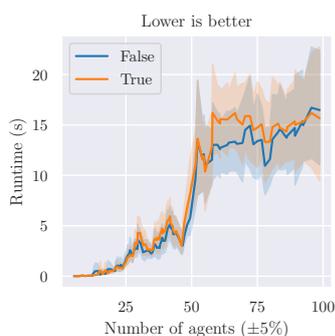


Figure 3.14.: The runtime performance of using the AllDifferent-constraint.

Min-Max-Modelling

The min max-objective is not directly available in CP-SAT and, thus, we have to implement it manually. We have two options: using $\Theta(|P|)$ linear

constraints

$$\begin{aligned} \min \quad & T \\ & T \geq t_p \quad \forall p \in P \\ & T \in \mathbb{R}_0^+ \end{aligned}$$

or using the non-linear `MaxEquality`-constraint of CP-SAT

$$\begin{aligned} \min \quad & T \\ & T = \max\{t_p \mid p \in P\} \\ & T \in \mathbb{R}_0^+. \end{aligned}$$

The first option require more constraints, but the second option is unnecessarily restricting and non-linear, i.e., potentially more complex. Because both options seem reasonable, we do a quick computational evaluation to find out which version is faster. Figure 3.15 shows that both options are nearly equally fast and solve the same percentage of instances to optimality. We use the second option as it results in a shorter formulation.

Multi-Threading

CP-SAT allows a concurrent execution on a specified number of threads. Every thread then performs a different strategy, as specified below.

1. The first thread performs the default search: The optimization problem is converted into a boolean satisfiability problem and solved with a Variable State Independent Decaying Sum (VSIDS) algorithm. A search heuristic introduces additional literals for branching when needed, by selecting an integer variable, a value and a branching direction. The model also gets linearized to some degree, and the corresponding LP gets (partially) solved with the (dual) Simplex-algorithm to support the satisfiability model. More details are given in Section 3.4.3 (How Does CP-SAT Work?) on page 65.
2. The second thread uses a fixed search if a decision strategy has been specified. Otherwise, it tries to follow the LP-branching on the linearized model.
3. The third thread uses *Pseudo-Cost branching*. This is a technique from mixed integer programming, where we branch on the variable that had the highest influence on the objective in prior branches. Of course, this only provides useful values after we have already performed some branches on the variable.
4. The fourth thread is like the first thread but without linear relaxation.
5. The fifth thread does the exact opposite and uses the default search but with maximal linear relaxation, i.e., also constraints that are more expensive to linearize are linearized. This can be computationally expensive but provides good lower bounds for some models. For our problem, the conditional constraints in Equations 3.13 to 3.15 will only be linearized here using big-M, but not in the default search.

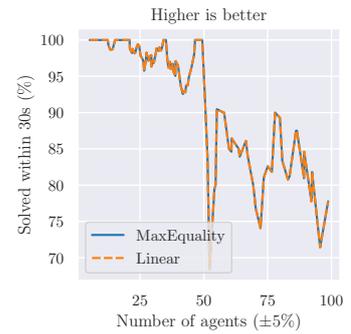


Figure 3.15. Performance of the two options to implement the min max-objective in CP-SAT. Both options perform equally well.

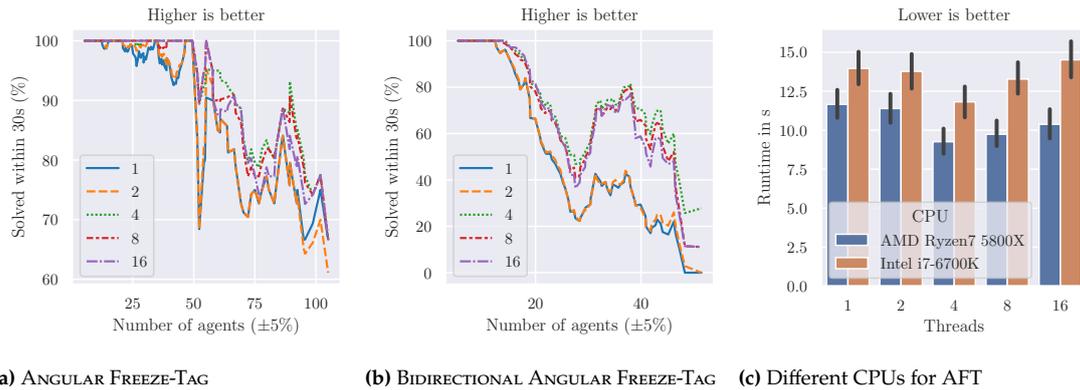


Figure 3.16.: Performance of different levels of parallelization for CP-SAT on (B)AFT. The best performance is achieved with 4 threads. This is also true for the more complicated bidirectional version and is independent of the CPU. The runtime is compared only on instances with 30 to 100 agents, and only if it was solved successfully on all CPUs as the comparison would get skewed otherwise.

6. The sixth thread performs a core based search from the SAT-community. This approach extracts unsatisfiable cores of the formula and is good for finding lower bounds.
7. All further threads perform a *Large Neighborhood Search (LNS)* for obtaining good solutions. More precisely, a *Relaxation Induced Neighborhood Search (RINS)* as proposed by Danna et al. [71] is used. This technique iteratively fixes variables in the LP-relaxation and tries to solve the (simpler) remaining problem.

There is a limited communication between the threads, such that new variable bounds are propagated but, otherwise, the threads run independently.

Gurobi, in comparison, does more actual work sharing among the cores as, e.g., node exploration in the branch and bound tree can easily be distributed among the cores. However, this is not possible for the particular expensive root node, and a surprisingly large amount of time has to be spent in the root node to make the branching actually effective. Gurobi simply solves the root node with different parameters on every core in the hope that some parameters perform better than others, but the results, e.g., cutting planes, are actually merged. When using the `ConcurrentMIP`-parameter, Gurobi launches multiple separate processes with different parameters for solving a MIP and, just like CP-SAT, simply returns the fastest*. Also the LP-solving utilizes a portfolio strategy: the first thread performs dual simplex, the fifth thread (if available) performs primal simplex, and all other threads work on a parallel barrier log algorithm. Generally, there is a visible speedup when using more cores but it is sub-linear[†]. Hence, Gurobi has more advanced parallelization capabilities than CP-SAT, but the simple portfolio strategy is still a very important aspect of its parallelization.

To see the influence of the parallelization, we executed CP-SAT with 1, 2, 4, 8, and 16 threads. The results in the left plot in Figure 3.16 are surprising, because more threads are not necessarily better even if the CPU has sufficient cores. Using only four threads, the best performance is achieved. This is also the case for the bidirectional problem variant

* https://www.gurobi.com/documentation/9.1/refman/concurrent_optimizer.html

[†] <https://www.gurobi.com/resource/parallelism-linear-mixed-integer-programming/>

(discussed in Section 3.7 (Adaptions for BAFT) on page 75) as can be seen in the plot in the middle. It does also not seem to be processor specific, as we have the same behavior on an Intel i7 6700K with four cores. We can also see that the old Intel CPU is only 27% slower than the new Ryzen CPU with eight cores.

Why are four threads faster than eight or sixteen threads? The first reason is likely to be that the performance of the first thread drops when there are many other threads competing for cache and memory. Also, the CPU-frequency drops faster due to the heat production of the other cores. If the other cores actually perform useful work, this is usually still a good trade-off. However, the second reason, which is relevant to our problem, deals with the fact that the fifth, sixth, and all further threads do not seem to contribute to solving the problem. Except of the sixth thread, they heavily rely on the linear relaxation, which is extremely weak for our formulation (as we know from the weak performance of the MIP-formulation).

Based on these observations, we use only four threads for further experiments.

3.4.3. How Does CP-SAT Work?

MIP-solvers like Gurobi or CPLEX have great documentations, and there are many books available that explain the underlying techniques (primarily Branch and Cut). However, there are unfortunately no such references for CP-SAT at the time of writing. As it is therefore impossible to refer to any useful resources to understand what is going on under the hood of CP-SAT (except of directly referring to the open source, but very advanced code), a short description is given here. Most implementation details of CP-SAT mentioned in this section are taken from a recorded talk by the developers [72], and the developers' comments in the official repository. It is quite possible that some of this information is already outdated or even wrong, but it should suffice to provide at least a basic understanding.

CP-SAT uses a SAT-solver to optimize its constraint programs, i.e., it converts them into boolean satisfiability formulas. This is not as simple and direct as the LP-relaxation utilized by MIP-solvers that 'only' needs to be integralized using a Branch and Cut-algorithm. By the famous Cook-Levin theorem [73], we know that we can express any problem in NP as a boolean satisfiability problem that itself can be solved by a SAT-solver. While for NP-hard optimization problems usually only the decision version regarding a bound is in NP, a binary search can also be used for optimization with satisfying formulas representing feasible solutions (upper bounds) and insatisfiable formulas providing lower bounds. Of course, the technique of Cook [73] is much too generic and would overstrain even the most powerful SAT-solver available, but direct encodings are much more efficient.

An integral variable can be encoded by a logarithmic number of boolean variables for each of its bits (assuming that we have fixed encoding sizes). The resulting encoding unfortunately allows only weak propagations. Instead, CP-SAT creates for every integral variable x and possible assignment v the boolean variables $[[x = v]]$ and $[[x \leq v]]$. The states $x < v$ can

be expressed by $[[x \leq v - 1]]$ and $x \geq v$ by $\neg[[x \leq v - 1]]$. This requires a potentially linear (on the domain!) amount of boolean variables, but the propagation for the constraints is stronger. If we have the integral variables x and y and the constraints $x + y = 3, x \geq 0, y \geq 0$, we can create the boolean clauses $(\neg[[x = 3]] \vee [[y = 0]]), (\neg[[x = 2]] \vee [[y = 1]]), \dots, (\neg[[y = 3]] \vee [[x = 0]]), (\neg[[y = 2]] \vee [[x = 1]]), \dots, (\neg[[x \leq 3]] \vee [[y = 0]]), (\neg[[x \leq 2]] \vee \neg[[y \leq 0]]), \dots$. Additionally, the boolean variables need to be made consistent, i.e., $[[x = 0]] \rightarrow \neg[[x = 1]], \dots$ and $[[x \leq 2]] \rightarrow [[x \leq 3]], \dots$. These are already quite a lot of clauses and variables for a very simple model, and it is unlikely that we could solve any larger model with this approach. However, most of the clauses are not actually needed, and could be added via lazy generation only when necessary.

Also, MIP-formulations can be exponentially large but still efficiently solvable by expanding only if necessary using callbacks. For example, the Dantzig-formulation of the TSP [58] has a theoretically exponential number of subtour-elimination constraints. Efficient implementations only add those constraints that are necessary whenever they stumble over a solution that is feasible in the current MIP-representation, but not in the fully expanded one. Of course, we need to create a separate, more powerful, problem model on which we can verify the solutions, and if there are violations, we need to efficiently find additional constraints to feed to the MIP-solver. Something similar happens in CP-SAT by only adding clauses and variables when needed. If the SAT-formulation is infeasible, so is the underlying model, which allows us to detect strong nogoods. If the SAT-formulation is feasible but the complete representation is not, it is correspondingly extended. The technique is known as Lazy Clause Generation, and is described by Ohrimenko et al. [74]. To improve the performance, the SAT-formulation can be reduced again; this is also done for larger MIPs, as smaller representations are faster to solve and prior extensions can become superfluous due to later extensions.

Solving Satisfiability Problems:

We now have converted our problem to a satisfiability formula, but how can we solve it efficiently? The basic technique used in modern SAT-solvers is actually surprisingly simple to understand, if one skips over the details not immediately relevant to basic function (e.g., efficient implementation, parameter tuning, supporting heuristics, etc.). We focus on the VSIDS-algorithm, which is the base for the SAT-solver in CP-SAT and the two predecessor algorithms, DPLL and CDCL. Many more details on these algorithms can be found in the books by Knuth [75] or Biere et al. [76] that focus exclusively on satisfiability (solvers). There is still a lot of development in SAT-solvers, but the *SAT Competition* (<http://www.satcompetition.org/>) gives a clue on what the currently best algorithm is.

The most naïve approach is to simply try out all 2^n assignments for a formula with n variables. We can improve this by assigning the variables one after the other and reverting our decision whenever we detect an infeasible clause, i.e., we assign the variables in a way that leaves all its

literals unsatisfied. This is generally known as backtracking and it is the most fundamental idea for most algorithms.

The *Davis-Putnam-Logemann-Loveland (DPLL) algorithm* is such a backtracking algorithm. It iteratively assigns a value to an unassigned variable and backtracks if it detects an infeasibility. Additionally, in each step, it uses *unit propagation* and *pure literal elimination* to speed up the process. For *unit propagation*, we check for each unsatisfied clause if there is only one literal remaining that could fulfill the clause and directly assign the corresponding variable. A simple example is the clause $(x_0 \vee x_1)$ with x_0 already assigned to `false`, which only allows to assign x_1 to `true`. For *pure literal elimination*, we check if an unassigned variable only appears in one form (negated or unnegated) in the still unsatisfied clauses, and we assign it correspondingly as it can do no harm.

The *conflict-driven clause learning (CDCL) algorithm* from the 1990s extends this idea by trying to add a clause that describes the core of the conflict whenever we need to backtrack. Additionally, we also backtrack directly to the root of the conflict instead of only one variable back. Adding such a conflict clause to the formulation does not change it; it is actually redundant, but it prevents us from performing the same mistake again with only slightly changed variables during further branching. If the clause in which we detected the conflict only consists of variables that have been directly assigned by branching, no useful conflict clause can be deduced. However, if some variables have been assigned by propagation (e.g., unit propagation or pure literal elimination), we can trace back which assignments lead to this and directly prohibit it by a clause. Consider the formula $(x_0 \vee x_1) \wedge (x_2 \vee x_3) \wedge (\bar{x}_0 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$. Let us assign $x_0 = \text{false}$, which results in $x_1 = \text{true}$ by unit propagation. If we now assign $x_2 = \text{false}$, we have to assign $x_3 = \text{true}$ by unit propagation, but this creates a conflict in $(\bar{x}_1 \vee x_2 \vee \bar{x}_3)$. The core of this conflict was setting $x_0 = x_2 = \text{false}$, and therefore we can add the clause $(x_0 \vee x_2)$. This will prevent us from creating this conflict again, or at least we can detect it earlier, even if this happens in completely different branches.

The *Variable State Independent Decaying Sum (VSIDS) algorithm* builds upon the CDCL-algorithm, but tries to branch early on conflict-prone variables. It was first used in the solver *Chaff* [77] in 2001, and is now implemented by most current CDCL-solvers including the one underlying CP-SAT. The VSIDS-algorithm associates every variable with a score, and every time we branch, we select the variable with the highest score. In the beginning, the score is simply the number of occurrences in clauses. Whenever we add a new conflict clause by the CDCL-algorithm, the score for the involved variables increases. To increase the influence of recent conflicts (or lower the influence of older conflicts), we periodically divide all scores by some constant.

Adding Linear Programming:

Mixed integer programming has some advantages for combinatorial optimization problems over classical SAT-solvers. The linear programming relaxations provide not only useful bounds due to duality theory, but additionally provide powerful cuts or good guesses for variable values in some cases. Even for the SATISFIABILITY problem, Cook et al. [78] showed

that cutting planes allow exponentially shorter proofs than resolution for some formulas. Constraints like *two out of N* can be easily expressed as linear constraint by $\sum_i x_i = 2$, but these need many clauses in a SATISFIABILITY formula. CP-SAT actually linearizes the constraint program to some degree and uses a (dual-) Simplex algorithm to (partially) solve it. Even cutting planes, e.g., Chvátal-Gomory cuts, are potentially added to improve the integrality. CP-SAT does not necessarily compute an optimal solution, but only performs a limited amount of steps (which can be helpful even if they do not lead to optimality due to duality). The results are used to detect infeasibility (which can be much more effective than resolution) and to obtain bounds on the objective and variables. Additionally, insights gained by the linear program can be used for branching decisions. At the default level, only some constraints are linearized. Constraints that are harder to linearize (including conditional constraints that require big-M) are only used at a higher linear relaxation level.

3.5. Heuristics

In the following section, we describe two heuristic approaches to compute non-optimal solutions. We already know an approximation algorithm which is guaranteed to yield solutions that are at most nine times worse than the optimum. While a factor of nine sounds already far from optimal, this is not the real issue, as this is usually just a generous upper bound and the actual solutions are often much better. However, the problem of this, and many other, approximation algorithms is that it is focused on the worst case instead of the average case. Simple local heuristics that focus on the average cases are, thus, also of high interest. We first consider a trivial random approach as a baseline, and then engineer a greedy algorithm.

3.5.1. Random

The most trivial approach to solve ANGULAR FREEZE-TAG is to iteratively choose a random feasible activation. This is easily possible for this problem as we cannot run into infeasibility. We simply start by choosing a random neighbor p_i of the start point p_0 to be activated. In the second step, we choose an arbitrary edge incident to the activated agents and perform the corresponding activation. This is repeated until all points are activated. To improve this most simple algorithm, we can run it n times and then choose the best solution.

Of course, one cannot expect great results from this algorithm, but it is generally interesting how much worse such an approach is. The better it performs, the simpler we can expect the corresponding instance to be. The random approach only has a reasonably good chance to yield a good solution, if the underlying structure of the instance is easy and bad decisions have a low influence.

When comparing the solutions of this random approach to optimal solutions computed using the constraint programming approach, we notice that the solutions are surprisingly good for a sufficient amount of

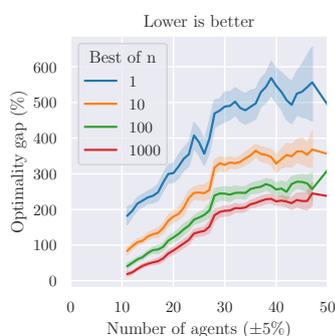


Figure 3.17.: Optimality gap of the best of n random solutions. We are only using instances we could solve to optimality such that the plot can be skewed for larger instances. The random instances are not as bad as one might suspect.

repetitions. The plot in Figure 3.17 shows that the best of 1000 random solutions has an average optimality gap of only 200%, i.e., the makespan is three times longer than the optimal solution. There are surely even better methods, but this is still closer than the quality guarantee of the approximation algorithm. As we are comparing against optimal solutions and we were not able to compute optimal solutions for all instances, this plot is potentially skewed toward the larger instances: it is possible that the optimality gap continues to increase, but the fact that we are only able to solve the simple instances with more agents dampens this effect.

3.5.2. Greedy

A greedy algorithm iteratively builds a solution out of partial solutions by choosing in each step the (locally) best extension to obtain a larger solution, until we obtain a complete solution. For our problem, this extension can be the next activation, but also the next k activations. When considering the next k activations, we also have the option to only perform the first activation. This potentially takes a longer runtime, but every activation is looking $k - 1$ activations into the future. When directly using all k activations, we only look on average less than $k/2$ activations into the future. The k generally has to be rather small, as it increases the solutions to consider exponentially and degenerates for $k = |P| - 1$ to a brute-force algorithm.

We have multiple options to select the best, next larger (partial) solution:

Makespan: The most straight forward objective is simply using the makespan of the partial solution. However, this potentially suffers from bad gradients, as only the last activation has any influence. Additionally, we should also try to reduce the other activation times in order to make future activations faster.

Sum: Alternatively, one can simply take the sum of activation times of all activated points in the partial solution. This will minimize the average activation time, which does not necessarily coincide with the minimal makespan but gives a better gradient as all activation times get minimized.

Squared Sum: Like the sum-objective but with squared activation times. This way, the longer activation times have a higher weight, and the focus is more on reducing the makespan than the average activation time.

Lexicographic: Order the activation times (decreasing) of already activated points and perform a simple lexicographic comparison to find the better solution. This approach can potentially suffer from numerical issues because a minimal deviation in one of the higher activation times can completely change the order. These small deviations can already appear by summing the float values in a different order.

When looking multiple steps ahead, the number of the solutions to consider can become rather large. We can utilize the fact that these objectives are monotonically increasing when solutions are extended. If we have a partial solution extended by k steps and one with worse objective that looks fewer steps ahead, we already know that it cannot compete with the previous solution and we do not need to extend it

further. In other words, we can cut off solution branches that cannot yield better solution extensions; this drastically decreases runtime. We simply save the best largest solution and discard any solution with a higher cost but not more activations. The size of a solution is only measured in the number of activated agents, but does not care about which agents are activated. This bounding solution is updated whenever a better one of the same size is found, or when we have a larger solution (which usually is an extension of the previous best and bounding solution).

We are now left with the question of which option yields the best solutions in which time. How much can we look ahead without increasing the runtime greatly, and how much does this actually help us? What are the differences of the four objectives?

If we only look ahead one step (i.e., activation) at a time, all objectives yield the same results with nearly the same runtime, and there is no decision to be made. Because every (greedy) activation is either free or increases the makespan, all four objectives choose the minimal extension of the makespan and, thus, perform the same steps.

Theorem 3.5.1 *For the greedy algorithm with $k = 1$, all four objectives return the same solution.*

Proof. With every activation added by the greedy algorithm, the makespan increases. All four objectives try to keep this increase minimal and therefore return the same result. Let us assume that the makespan does not increase and the activation requires a rotation. If it does not require a rotation, none of the objectives would increase and it would be minimal for all of them. If it requires a rotation, only points with an activation time lower than the makespan can be the activator, as it would otherwise increase the makespan. However, if this activation does not increase the makespan, it would have been available and at a lower cost for all four objectives before, contradicting the greedy selection strategy. \square

For a look-ahead depth of $k \geq 2$, there can be differences, but which objective returns the best results and which allows better pruning? To answer this question we compare the performance and runtime of the different objectives at different look-ahead depths in Figure 3.18. We keep things simple and limit ourselves to single-step moves that are potentially slower but better for now.

The runtime is strongly influenced by the depth and the objective with the sum-objective performing by far the worst. This indicates that it cannot prune as well and as early as the other objectives. A shorter partial solution tends to have a smaller sum even if it is generally worse. It has a higher average activation time, and needs additional arguing that it cannot be better than the longer solution or has to be branched. However, also the squared sum-objective sums the values up. Why can it still prune more than the mere sum? The reason for this can be that squaring the values increases the penalty of bad decisions. If a bad rotation is performed in an early branch, it has a stronger influence when squaring the value instead of just summing up, and thus hits the upper bound earlier.

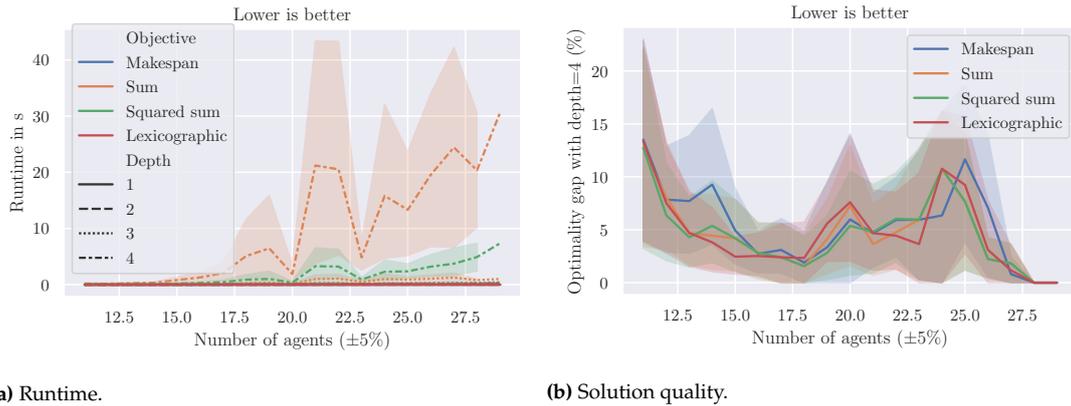


Figure 3.18.: Comparison of the runtime and performance for the greedy algorithm with different objectives and look-ahead depths. The performance (measured in the optimality gap) is only displayed for a look-ahead depth of 4. Depth and objective have both a strong influence on the runtime. The solution quality of all objectives is comparable with an average optimality gap of 5% to 10%.

While the objective has a strong influence on the runtime, it has surprisingly little influence on the solution quality. The solution quality is again measured by comparing to the optimal solutions provided by the constraint program. As before, this can skew the results for larger instances as not all instances could be solved to optimality. For the instances which could be solved to optimality, the average optimality gap is between 5% to 10%.

To consider larger instances, depths, and the influence of single-steps, we focus on the fastest objective of the previous experiment, the lexicographic comparison.

We now want to validate our previous decision to choose single-steps. In Figure 3.19, we see that performing only the first activation toward the best found expanded solution yields indeed slightly better results. While the optimality gap is visibly reduced, the overall improvement is relatively small because both options produce good solutions. As the runtime increases only linearly with the look-ahead depth, it is recommendable to perform single steps at least for smaller look-ahead depths.

One important remaining question is the influence of the look-ahead depth on the solution quality, which we try to answer with the plots in Figure 3.21. First, we can see that also the runtime with the lexicographic-objective increases strongly with a higher look-ahead depth, and that instances with 300 or more agents can already take many minutes for a depth of 3. A depth of more than 6 will quickly become prohibitively expensive, at least for larger instances, such that there is no advantage to the relatively efficient constraint program. The solution quality continues to increase with higher look-ahead depth. However, even at depth 6, it does not reach the quality of the solutions of the constraint program with a comparable runtime. Still, the greedy algorithm performs reasonably well even for low depths. It is important to note that the optimality gap is this time measured on lower bounds, and not on optimal solutions which are not available for such large instances. The reason why the optimality gap increases can simply be due to a worse lower bound and not a worse solution of the greedy algorithm, but this allows us to see that even for large instances, the optimality gap is at most around 20%.

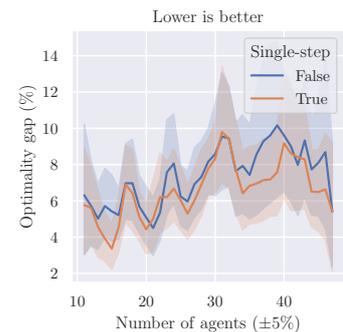


Figure 3.19.: Performance of single-step vs. applying the full look-ahead for the greedy algorithm with a depth of 5.

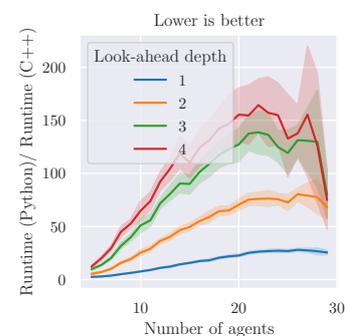
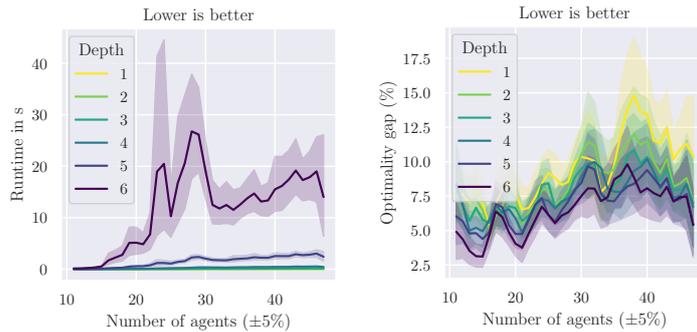
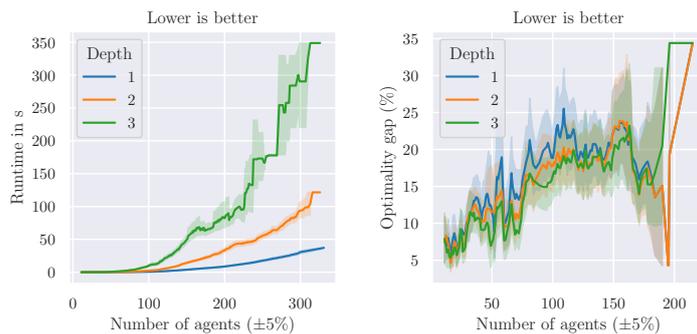


Figure 3.20.: Runtime comparison of a C++ vs. a Python-implementation of the greedy algorithm. The C++-implementation is up to 200 times faster, even for larger instances with multiple seconds runtime.



(a) Runtime.

(b) Solution quality.



(c) Runtime (large).

(d) Solution quality (large).

Figure 3.21: Runtime and solution quality for the greedy algorithm with lexicographic objective and single-steps. The runtime is strongly increased by higher look-ahead depths, but also the solution quality improves visibly. The optimality gap is computed based on the lower bound returned by CP-SAT and not on optimal solutions to extend the range of instances. For larger instances, no lower bound is available.

We implemented the greedy algorithm, contrary to the other algorithms, in C++. The importance of an efficient programming language becomes apparent when comparing the runtime of a Python-implementation with the runtime of a C++-implementation in Figure 3.20. Especially for higher depth, for which a lot of partial solutions have to be quickly created and compared, the C++-implementation is around 100 to 200 times faster. This can make the Python-implementation even slower than CP-SAT. The use of a profiler indicates that a lot of the time is spent in branching solutions for which a solution has to be copied to apply different activations. By utilizing more efficient solution representations in, e.g., NumPy, we could possibly also speed up the Python-implementation significantly, but it is unlikely to come close to the runtime of a C++-implementation.

3.6. Evaluation

In the previous sections, we have seen a number of individual benchmarks for specific algorithms and approaches. In this section, we try to put them all into comparison. We start by describing the setup and the used instances (that have also been used for the previous smaller experiments), then compare the two approaches with optimal solutions, and finally we try to get good solutions for very large instances using all approaches.

3.6.1. Benchmark: Instances, Software, and Hardware

We re-used the same instances as for `MINIMUM SCAN COVER`, see Section 2.6, but increased the range of vertices and edges. Additionally, we added a uniformly random initial heading to each agent. The distribution of the increased instance set can be seen in Figure 3.22.

The instances have been solved on desktop workstations equipped with AMD Ryzen 7 5800X (8×3.8 GHz) and 128 GB of RAM. The mixed integer programs were solved with Gurobi 9.1.2 and the constraint programs were solved with Google's CP-SAT in ortools 9.0.9048. Except for the greedy algorithm, the code has been implemented in Python 3.8.8. The greedy algorithm has been implemented in C++20 with a Python-interface because the pure Python-implementation was not satisfactorily efficient. The time limits differ for various experiments, and the modelling time of the mixed integer program and the constraint program are not considered. The reason for this is that the modelling process is not as optimized as it could be. However, we still abort the modelling process if it takes too long. For smaller time limits, CP-SAT can also be inaccurate and actually abort earlier, e.g., after 8 s instead 10 s.

3.6.2. Optimal Solutions

During the tuning process of the mixed integer program and the constraint program, we already saw that the constraint program is much more efficient than the mixed integer program. We now want to analyze which instances can actually still be solved to provable optimality. The instances were solved with a time limit of 30 s, and there was only one trial per instance because Figure 3.23 shows that the runtime of CP-SAT is reasonably deterministic.

First, we can see in Figure 3.25 that the mixed integer program cannot even solve instances with 20 agents. It cannot even prove any usable lower bounds for instance with more than 25 agents, as can be seen in Figure 3.24.

The constraint program, on the other hand, even solves instances with 120 agents to optimality by a chance of 50%. The celestial instances seem to be harder to solve while for the general instances, even around 50% of the instances with 150 agents have been solved to optimality. Compared to the other optimization problems in this thesis, the decline of solved instances is surprisingly slow but early. We encounter a lot of hard instances early, which slowly increase until we are only able to solve around 60% of the instances before we have a heavy drop. At this point, the corresponding formulations already reached a considerably large number of variables and constraints which simply become hard to handle within a time limit of only 30 s. This indicates that it is probably less the size of the instances but specific properties of them that make the instances hard. When looking on the distribution of the solved instances, we can only see a small tendency of sparse instances to be slightly easier, but they also have smaller formulations. This leaves us with the observation that the `ANGULAR FREEZE-TAG PROBLEM` is surprisingly easy to solve to optimality for many instances, but that there are also many smaller instances that

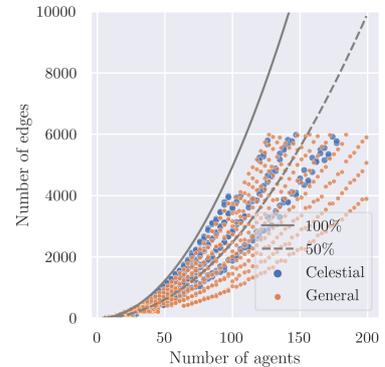


Figure 3.22.: Instance distribution.

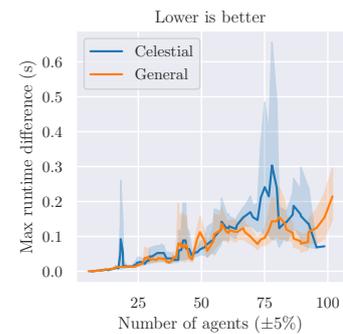


Figure 3.23.: The maximum difference of the runtime of CP-SAT for solving an instance 10 times with a time limit of 30 s. In nearly all cases, either all trials solved the instance to optimality or none. This indicates that the performance of CP-SAT is reasonably deterministic and has only a small variance.

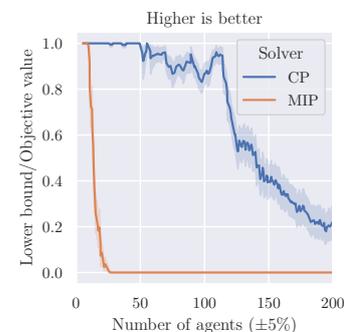


Figure 3.24.: The lower bounds of the MIP and the CP. The MIP is quickly incapable of providing any useful lower bound while the lower bound quality of the CP only declines slowly.

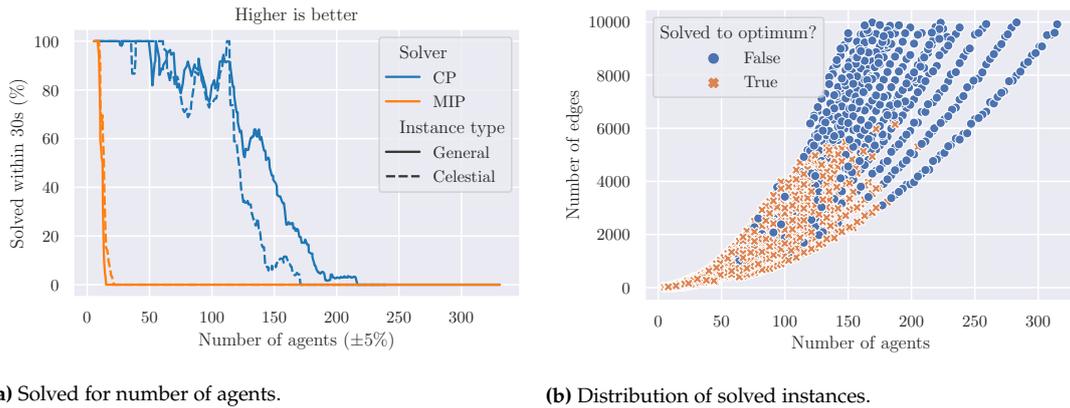


Figure 3.25.: Instances solved to optimality within 30 s in percent and the distribution over the amount of edges and agents. The MIP performs poorly, while CP-SAT can solve larger instances to provable optimality. The distribution shows that sparse instances have a slightly higher chance of being solved to optimality but that is barely notable.

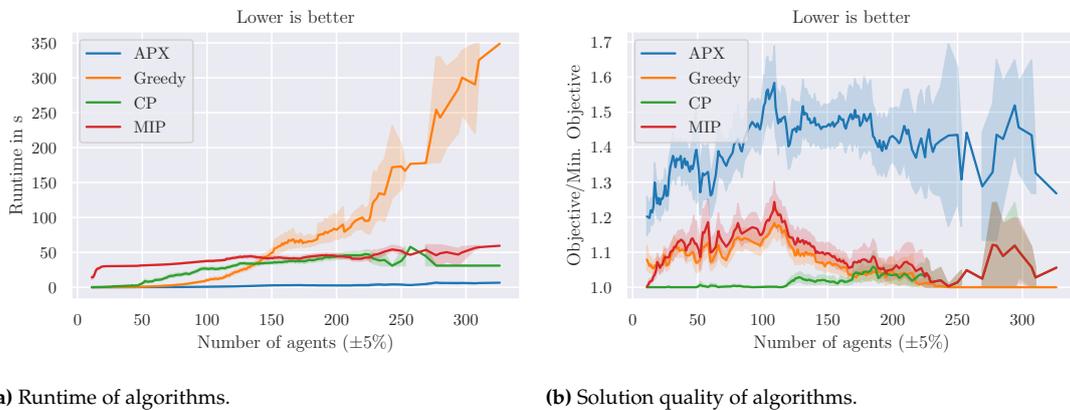


Figure 3.26.: Runtime and solution quality (objective value divided by best known objective value) for different algorithms. The greedy algorithm uses a look-ahead depth of 3. The CP and MIP return a greedy solution with depth 1 if they are not able to compute a solution.

are very difficult, fitting to the NP-hardness of the optimization problem.

3.6.3. Good Solutions

We can compute optimal solutions for many instances, but what can we do for the remaining instances? Our only option is to use approximative or heuristic approaches to obtain at least reasonably good solutions. Thus far, in Sections 3.4 and 3.5, we were introduced to multiple approaches: an approximation, a random approach, a mixed integer program and a constraint program (which can also return intermediate solutions at time-out), and a greedy algorithm. The random approach can be discarded as it has no chance of competing against the other approaches, as we have already seen in the preliminary experiments. We compare the solutions of the approximation algorithm, the exact algorithms with their intermediate solution or a simple greedy solution of depth 1 at timeout, and the greedy algorithm with a look-ahead depth of 3. As we do not have optimal solutions or even reasonably good lower bounds for all instances, we compare the performance in relation to the best performance: for every instance, we use the best solution of any approach as baseline.

The plots in Figure 3.26 show that the approximation algorithm is very fast and also reasonably good. Most of the time, its makespans are less than 50 % longer than the best known solution (on average 38 %). As expected, the constraint program leads the pack for smaller to medium sized instances. Only at around 150 agents, the greedy algorithm becomes competitive to the constraint program. Overall, the greedy algorithm yields good solutions at a good scalability. In combination with the constraint program, which needs an initial solution, it can solve most instances to optimality or at least with an average optimality gap of less than 20 %. For very large instances, the approximation algorithm yields reasonably good solutions in a fraction of the time.

3.7. Adaptions for Bidirectional-AFT

Until now, we have just considered one-sided adjustments, but for practice we are more interested in two-sided adjustments (both sides must point toward each other). We can adapt our approaches from the one-sided variant to the two-sided variant. In the following, we describe how to perform this adaption for the constraint program and the greedy algorithm, including some preliminary performance evaluations.

3.7.1. Adapting the Constraint Program

The constraint program can easily be adapted to require the receiver to adjust to the sender. Currently, only the sender will point to the receiver, but the receiver will still head toward its starting position at this time. However, the time dependencies for activating neighbors is nearly identical to being activated. This allows us to adapt the previous constraint program with only some small changes.

First, we remove the constraints in Equation 3.14, as agents will no longer head toward their initial heading at the time of their activation.

Second, the activation time needs to allow the activated agent to rotate from its initial heading toward the activating agent. More precisely: if p activates p' , $t_{p'}$ has to be at least the time p' needs to rotate from its initial heading to p .

$$x_{p \rightarrow p'} \Rightarrow t_{p'} \geq \alpha(p' s_{p'}, p' p) \quad \forall p \in P, p' \in N(p) - p_0 \quad (3.18)$$

Third, we have to integrate the activating agent in the rotation time dependencies that are currently enforced by the constraints in Equation 3.15. More precisely: if p activates p' and p' activates p'' , then $t_{p''}$ has to be at least $t_{p'}$ plus the rotation time of p' from p to p'' .

$$\begin{aligned} \forall p \in P, p' \in N(p) - p_0, p'' \in N(p') - p_0, p \neq p'' : \\ x_{p \rightarrow p'} \wedge x_{p' \rightarrow p''} \Rightarrow t_{p''} \geq t_{p'} + \alpha(p' p, p' p'') \end{aligned} \quad (3.19)$$

For $p' = p_0$, we need

$$x_{p_0 \rightarrow p} \Rightarrow t_p \geq \alpha(p_0 s_{p_0}, p_0 p') \quad \forall p \in N(p_0) \quad (3.20)$$

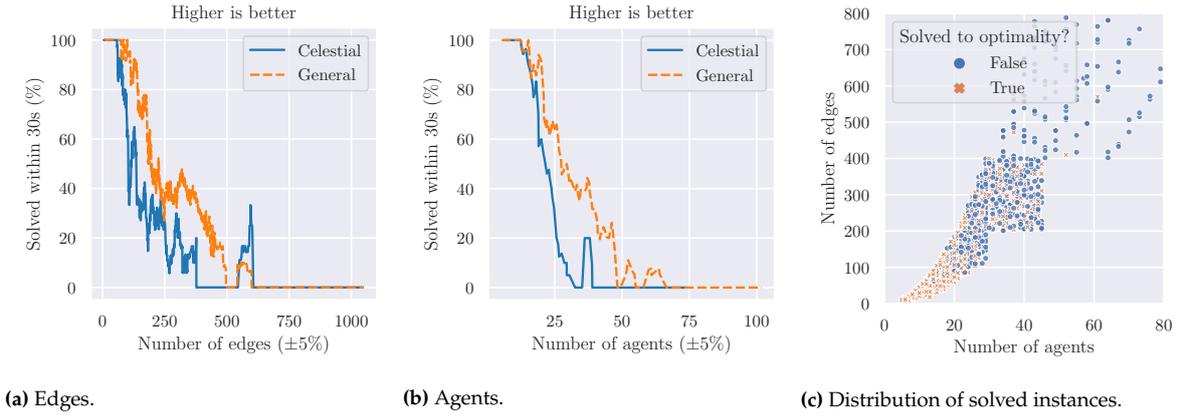


Figure 3.27.: Percentage of BAFT instances solved to optimality over size (number of edges and agents). Additionally, the distribution shows that the density of the graph does not seem to have an influence. We can only solve much smaller instances of BAFT than of AFT. The general instances are slightly easier than the celestial ones.

because p_0 does not need to be activated but can directly rotate from its initial heading, like in the original problem variant.

These new constraints now enforce that every agent has enough time to rotate from its initial heading before being activated, and that agents can rotate toward their activating agent before activating any other agents. Due to the triangle inequality, it is sufficient to make sure that the activation times are far enough apart and we do not need to encode the concrete order. The only explicitly enforced part of the rotation order is in Equation 3.19 by not using absolute values as in Equation 3.15. Note that Equation 3.15 is still used to ensure enough rotation time between consecutive activations.

The performance of this formulation can be seen in Figure 3.27. For 20 agents, instances already become hard, especially for celestial instances. For general instances, we can still solve instances with less than 30 agents with a reasonable (50%) chance, but no instance with more than 70 agents could be solved to optimality anymore. The edge density does not seem to influence the hardness. While the instances also get harder with more edges, this seems to be primarily due to the correlation with the number of agents.

3.7.2. Greedy Algorithm

The adaption of the greedy algorithm is trivial: it actually just optimizes the activation edges based on the computed activation times. Therefore, we only have to adapt the way the activation times are computed.

The performance of the greedy algorithms is comparable to our original problem, AFT, or possibly even better as our lower bounds are not as strong. This is shown in the plots of Figure 3.28. The objectives have only little influence on the solution quality but a strong influence on the runtime. The optimality gap using the lexicographic objective decreases with the look-ahead depth notably but not strongly, as it is already good at low depths. For a look-ahead depth of 1 to 6, the optimality gaps are 12.82%, 11.48%, 10.55%, 9.38%, 8.55%, and 7.62%. The difference decreases for larger instances.

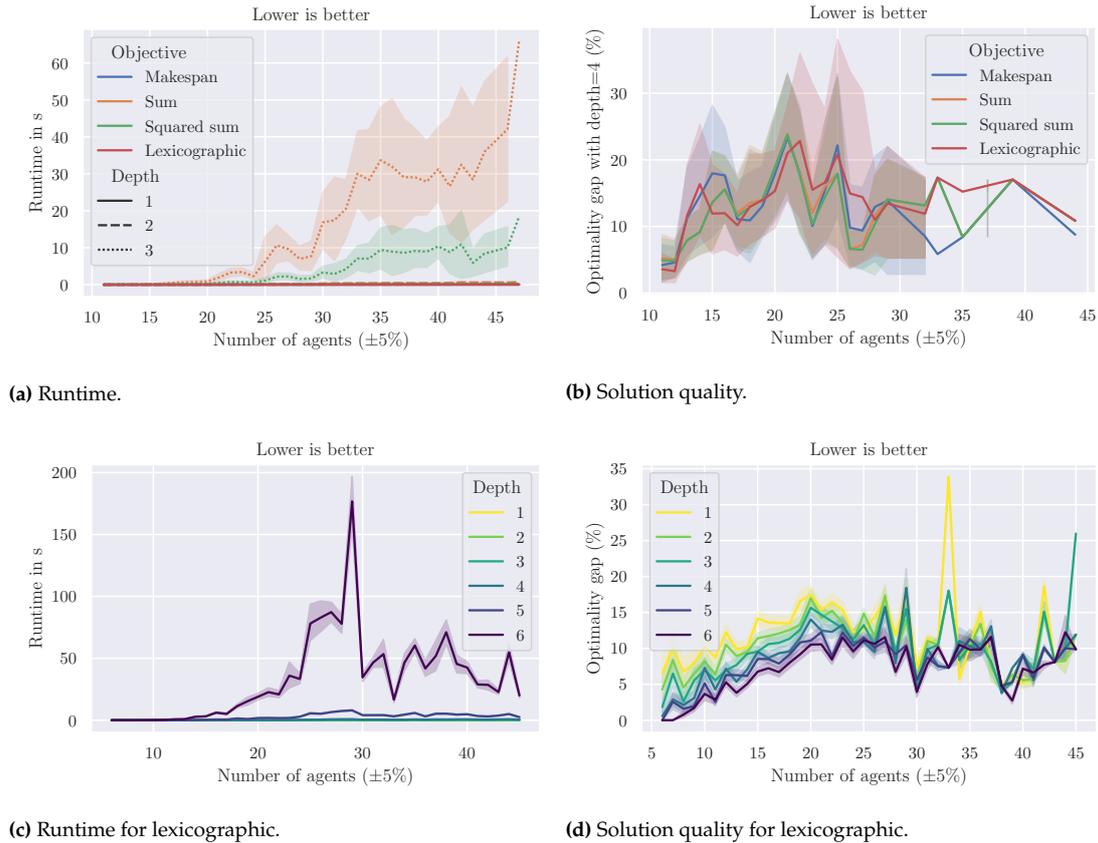


Figure 3.28: Performance of the greedy algorithms for BAFT. The objectives show the same runtime differences as for AFT, with the sum-objectives being the slowest by far, and the lexicographic objective being the fastest. The performance of the four objectives with a look-ahead depth of three is nearly identical, with the lexicographic-objective being minimally better. By increasing the look-ahead depth, we can improve the optimality gap at the cost of a higher runtime.

3.8. Conclusion

This chapter considered the **ANGULAR FREEZE-TAG PROBLEM (AFT)** and, the more practical variant, the **BIDIRECTIONAL ANGULAR FREEZE-TAG PROBLEM (BAFT)**. We focused on solving instances to optimality and did a deeper dive into the behavior of CP-SAT. This showed not only that CP-SAT can deal well with numerical variables, but also that increasing the resolution results in an only slightly-increased runtime, despite the need to convert every numeric variable into many boolean variables. Many instances of AFT show to be benign, and CP-SAT is able to solve many instances with over 100 agents to optimality within seconds. On the other hand, there are also many smaller instances which show to be hard at least for CP-SAT. Mixed integer programming performed poorly on this type of problem, as it is not able to prove good lower bounds, probably due to the many applications of the big-M-method. Also, the approximation algorithm and the greedy algorithm return reasonably good solutions with average optimality gaps below 50% resp. 20%. For BAFT, only a small adaption of the CP and the greedy algorithm are necessary, but the instances prove to be harder to solve to optimality. Only instances with less than 25 agents have a good chance of being solved to optimality. The greedy algorithms perform well, comparable to AFT.

Automated Data Retrieval from Large-Scale Distributed Satellite Systems*

4.

This chapter presents a distributed auction-based scheduling approach for maximizing the value of the downloaded data from a satellite swarm. The method allows competing satellite operators to bid for contact times and has a fair and transparent price estimation based on the competition. On its own, it can also be used with a simple bidding strategy to obtain good schedules; this is demonstrated on benchmark simulations with up to 1080 satellites. As a consequence, we are able to achieve value rates of 74 % of the available data, compared to just 28 % for standard greedy strategies.

4.1. Introduction

The previous two chapters focused on intersatellite communication that is advancing but not common, yet. Currently, centralized communication with ground stations (such as in Figure 4.1) is used for most constellations. In this chapter, we consider a central challenge that is mission critical for the successful operation of large-scale satellite constellations in Low-Earth Orbit with ground stations: How can we coordinate the short-term download operations for the enormous amounts of generated data, based on wireless line-of-sight connections to a limited number of stationary ground stations? These issues are critical for the future growth of space systems, with multiple commercial space operators competing for downloading their commercial data in a timely fashion, relying on the services of a scarce set of ground stations that is subject to numerous strong constraints, so it cannot simply be expanded.

The trend of using distributed space systems – such as satellite constellations – instead of monolithic systems has been growing for the last decade. Some of the newly announced constellations feature more than 1000 satellites. Traditional spacecraft operations involve manual control of the spacecraft by skilled human operators, following a 4-eyes principle. Even when operators batch multiple telecommands together, this process is still time-consuming and prone to errors. In addition, the autonomy level aboard traditional spacecraft is low; critical activities, such as anomaly identification and resolution, must be supervised by the operators. This manual approach is not (linearly) scalable to large satellite constellations, resulting in a variety of algorithmic challenges for automated operation. According to the NASA Technology Roadmap [79]: “Demand continues for ground systems which will plan more spacecraft activities with fewer commanding errors, provide scientists and engineers with more functionality, process and manage larger and more complex data more quickly, all while requiring fewer people to develop, deploy, operate and maintain

* The content of this chapter has been presented at CASE 2019 [4] and was spawned by the ASIMOV project at the European Space Agency. Other papers of this project include [15–17]. Many thanks to the project members Mohamed Khalil Ben-Larbi, Mirue Choi, Jonas Dippel, Sándor P. Fekete, Benjamin Grzesik, Andreas Haas, Tom Haylok, Harald Konstanski, Michael Perk, Kattia Flores Pozo, Volker Schaus, Christian Schurig, and Enrico Stoll for their collaboration.

4.1	Introduction	79
4.2	Related Work	81
4.2.1	Decentralized Auction- Based Scheduling	82
4.3	Preliminaries	82
4.4	Concept	83
4.5	Auctioneer’s Algorithm	84
4.5.1	Optimal Bid Selection	85
4.5.2	Price Estimation	85
4.6	Bidding Strategy	87
4.7	Experimental Evaluation	87
4.7.1	Greedy Algorithm	88
4.7.2	Experimental Results	88
4.7.3	Contact Pauses	89
4.8	Conclusion	89



Figure 4.1: A ground station for satellite communication (Galileo IOT L-band antenna). Copyright by European Space Agency (ESA/C. Lezy,CC BY-SA 3.0 IGO).

them". This gives rise to numerous problems of robotics and automation, many of which involve extremely complex systems and requirements.

One of the critical aspects of satellite constellation operation is being able to promptly retrieve the generated payload data. For satellites in low-earth orbit (LEO), this data is typically based on visual imaging, with total size growing quadratically with increasing image resolution and linearly with time resolution; in fact, the demand for up-to-date, fine-grained images is one of the driving forces behind the rapidly expanding number of satellites in current and projected constellations. Downloading this data relies on wireless communication between the orbiting satellites and a limited number of stationary ground stations with powerful parabolic antennas, requiring a direct line of sight; note that even when the underlying metadata (describing size and importance of payload data consisting of image files) is relatively small (and thus rapidly communicated), the download times for the actual payload data itself are considerable. As a consequence, these communication links become a critical bottleneck for successful, timely operation of a satellite constellation. However, coordinating these communications for massive satellite constellations and enormous amounts of data involves excruciatingly difficult optimization problems.

Currently, most satellite systems have their own dedicated ground stations. However, the demand for growing the number of satellites (e.g., for providing high-resolution, short-term imaging data) is quite high, whereas the building and operation of ground station (which should be located at high geographic latitudes, i.e., close to the poles in order to provide frequent communication windows) is subject to numerous geographic, logistic and political constraints, as well as quite expensive. As a consequence, it is foreseeable that ground stations will have to be shared, enabling these systems to be more dynamic and deal with temporary bandwidth bottlenecks while also lowering the prerequisites of new satellite systems. Furthermore, outsourcing the maintenance of the ground stations to a *Ground Station as a Service* will allow satellite operators to focus on the satellites themselves. While there are already sophisticated scheduling algorithms for prioritizing important data in the presence of limited bandwidth (as described in the section on related work), they miss the element of a bidding and pricing scheme for competing satellite systems needed for such a service.

In this chapter, we present an auction-based approach for coordinating and prioritizing download activities of a satellite constellation fit for a *Ground Station as a Service* system. In it, satellite operators bid on communication slots of ground stations and are billed correspondingly. If the occupancy is high, the prices automatically rise such that important, i.e., high-valued data is prioritized. We also demonstrate the power of our approach on benchmark constellations with more than 1000 satellites and a time window of 36 days: Our auction-based approach manages to retrieve ~74% of the total data valuation, compared to merely ~28% that are achieved by a standard greedy-type approach that is typically used by human operators with automation assistance.

4.2. Related Work

Scheduling is a common problem in computer science with a vast amount of research and literature. A thorough introduction to the topic is given by Pinedo [80]. In addition to scheduling theory in general, there has been a considerable body of work on scheduling in the context of satellites. In her PhD thesis, Spangelo [81] considers operational challenges specific to small satellites in LEO, such as restricted on-board energy and data storage capacity.

Unfortunately, finding an optimal solution for even very restricted offline versions of the scheduling problem is known to be NP-hard. Therefore, exact solutions do not scale to large constellation sizes. However, for practical purposes, an algorithm that always achieves an optimal schedule is often not needed, motivating approximations algorithms and heuristics that are considered in the literature.

Scheduling problems that arise in practice are often inherently online in nature, i.e., scheduling decisions must be made without complete information. This also holds true for satellite operations: not all customer requests are known ahead of time, so it may be desirable to change the schedule after observing an event of interest. Furthermore, anomalies may occur during operation. Li et al. [45] consider an online scheduling variant with stochastic arrivals of urgent tasks and sequence-dependent setup times. They make a distinction between normal and urgent tasks. Rescheduling during orbit (between ground contacts) requires autonomous decision-making with limited/local information by the satellite. This is especially true if rescheduling is triggered because the satellite itself observed/measured an event of interest. Urgent customer request may be relayed via inter-satellite communication links (if available) from ground station to satellite. Stottler [82] reports a prototype implementation for a scheduler that incorporates case-based reasoning to automatically resolve conflicts. This conflict resolution is based on past decision-making and approval of human schedulers. A method for scheduling services in large satellite constellations is described by Marinelli et al. [83], in which types of scheduling problems are identified, and a time-indexed integer programming formulation is used. They also describe a practical heuristic approach based on Lagrangian relaxation and a Fix-and-Relax algorithm. The method was experimentally evaluated with promising results on the GALILEO project in research between Telespazio and the European Space Agency. Augenstein et al. [46] describe the scheduling algorithm that is used for the Terra Bella constellation. They consider the following problem setting and constraints: agile satellites that can change orientation, non-zero setup times for ground stations between different satellite contacts, a required minimum contact frequency for each satellite to transmit health/telemetry data and the ability for human operators to manually “lock-in” or “lock-out” a given contact. The objective is to balance image collection and data downlinking time, while maximizing the image collection of priority-weighted targets. Lee et al. [84] show a genetic algorithm for scheduling, which is designed with regard to the Korea Multi-Purpose Satellite (KOMPSAT) series.

4.2.1. Decentralized Auction-Based Scheduling

As an alternative to centralized scheduling, Wellman [85] proposed *auction-based decentralized scheduling*, with competing agents bidding for resources. The schedule and the prices for the resources are determined by auction or market mechanisms. Prices may be purely virtual, but can also be used to determine actual usage fees. Agents work for their own advantage and may hold information private regarding their strategy. They have to evaluate the trade-offs of acquiring resources to maximize their objectives potentially with a limited budget.

In our case, satellites bid for communication slots of ground stations trying to maximize their downlinked data value while minimizing the overall costs. Therefore, the ground stations act as auctioneers. As in other auction-based mechanisms, the auctioneer continuously posts the price quotes to the agents. The agents communicate their bids iteratively, so that they can react to competing parties. After a fixed time window, the auctioneer computes the final schedule and price.

An auction may be differentiated across many parameters such as price determination algorithm, event timing, bid restriction, and intermediate price revelation. One of the most important distinctions is whether an individual auction allocates a single resource or several resources at once (*combinatorial auctions*).

The term *combinatorial auction* was popularized by Rasseneti et al. [86] in 1982. Later work on the complexity of the problem [87] showed that it is NP-hard to determine an allocation once all bids have been submitted to the auctioneer. These results lead to further research on the topic [88]. Solving distributed scheduling problems with market mechanisms has been proposed multiple times [89, 90]. Wellman [91] named his approach *market-oriented programming* (MOP).

Prior work has successfully applied market-inspired mechanisms to scheduling [92, 93] and other distributed resource allocation problems. Several have adopted the framework of general equilibrium theory and have found that the computational markets behave predictably when the conditions of the theory are met [91]. Additionally, the MOP approach was applied to a variety of discrete optimization problems. It has been successfully adopted to scheduling problems in manufacturing [94–96] and train or airport slot allocation [97–99]. Moreover, the benefits of the approach have been widely used in transportation services [88, 100].

Depending on the application of the scheduling problem, one can investigate bidding strategies that produce better solutions [101]. Bidding strategies like *iBundle* [102] (in which the price update is based on bid prices from unsuccessful agents) have brought great success to train scheduling problems [97].

4.3. Preliminaries

Given the difficult conditions (with enormous distances and large amounts of data), ground stations and satellites use directed antennas for high-bandwidth communication. This requires satellites not only

to be in range, but also necessitates precise adjustments of both sides, which can take up to two minutes. This also implies that satellites and ground stations have to be informed in advance of scheduled contacts. Because satellites in LEO are moving very fast (they can orbit earth in 90 minutes), the feasible contact windows are only a few minutes long. In addition, many satellite constellations operate in sun-synchronous orbits. As a consequence, satellite distribution is heterogeneous, with particularly high density near the poles, as polar orbits and ground stations increase the frequency of contact windows.

This leads to a basic model with the following underlying assumptions; some of them constitute slight but legitimate simplifications that are used in the following experiments.

- ▶ Ground station and satellites can establish a connection if the line of sight is not interrupted and a pass can be reliably predicted. Hence, for every satellite and ground station there is a continuously extended list of feasible contact windows (each having a minimal start and a maximal end time) in which data could be transmitted if there are no conflicts (see next point).
- ▶ Satellites and ground stations each need 2 minutes of adjustment time before each scheduled contact. During adjustment and contact, no other contacts can be performed.
- ▶ For simplicity, we use a homogeneous bandwidth. The amount of downloaded data hence can be measured in contact time.
- ▶ The onboard storage of each satellite is limited to the amount that could be downloaded in 10 000 seconds. If the memory is full, the least valuable data is automatically deleted. Data is created randomly (uniformly distributed size, value, and time).
- ▶ A contact can be performed if it is scheduled at least three hours in advance.
- ▶ The satellite operator has a knowledge of the data value on board of the satellites.

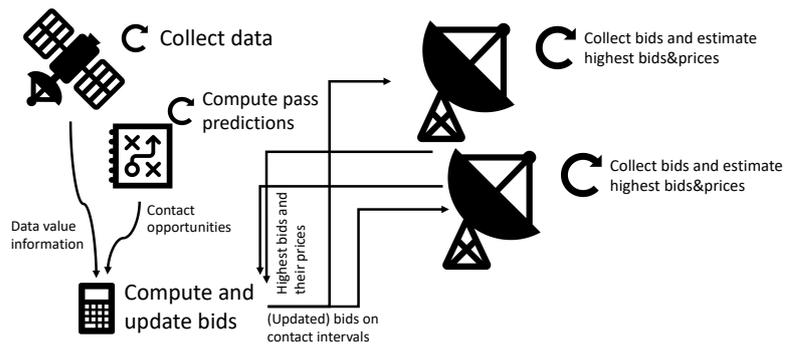
The last two assumptions are non-trivial but can be implemented by an additional low-bandwidth connection which are less restrictive than high-bandwidth connections and sufficient since the corresponding data is much smaller than the actual payload. But even without such an additional connection, data value can to some degree be predicted and supported by piggybacking informations during other contacts. Some satellites also do not need to be informed about contacts in advance because they automatically anticipate data from ground stations in reach and adjust correspondingly to some ground station.

To evaluate the scheduling quality, we consider in our experiments the objective of data and value rate as well as minimizing contact pauses. The data rate is the percentage downloaded of all generated data. The value rate is analogous but considers the value of the data.

4.4. Concept

In our algorithm (see Figure 4.2), each satellite acts as a bidder who wants to buy contact times on ground stations to maximize the value of its downlinked data minus the price paid for the contacts. Ground stations

Figure 4.2.: Based on data value estimations, and pass predictions, bids on contact intervals for each satellite are computed and sent to the corresponding ground stations. The ground stations constantly collect these bids and determine the highest bids and their current prices. Based on this information, the satellites can update their unsuccessful bids.



act as auctioneers that accept bids on usage intervals and continuously determine the currently highest bids and their prices. A bid wins the auction when it remains the highest bid until a specified time before the beginning of its interval, such that there is enough time to actually schedule the contact. The currently highest bids are broadcast frequently, so satellites can move their bids into intervals with lower prices. All bids that are not marked as highest bid can be retracted or updated. The prices have to be fair and based on the actual demand (i.e., competition) such that they can be used as an actual fee in a *Ground Station as a Service* system. Note that neither the satellite nor the ground station will actually do any computations; these are carried out by separate computers, as the satellites do not have the required computational and communicational capabilities. Hence, the data value estimation for the satellites that is important to determine the bid amount is only based on meta-information and predictions.

There are two primary components:

1. An auctioneer's algorithm that determines the highest bids and their prices. It needs to be fast for continuous evaluation and transparent to prove its fairness.
2. A bidding strategy for satellites that can be different for each satellite.

While centralized scheduling algorithms like the one of Lee et al. [84] can be used to obtain a comparable (possibly even better) schedule, they miss a pricing scheme and do not allow different dynamic bidding strategies for competing satellite operators. Our approach provides not only a good scheduling quality, but also a fair pricing scheme and individual control for each satellite operator.

4.5. Auctioneer's Algorithm

The auctioneer has to continuously select the winning bids for time slots as well as the corresponding prices.

4.5.1. Optimal Bid Selection

If a ground station is given a set of offers for intervals, it is not obvious how to select the highest bidders. While for independent resources one could simply always take the highest offer, this is not the case if the resources are overlapping as it is the case for time intervals: Multiple short intervals with low offers can together outweigh a high offer for a long interval. While one could still accept the interval with the highest offer or the highest relative offer (price/length), either strategy would encourage to either buy very long intervals or to drive out the competition by very short intervals. The most reasonable approach is to select the intervals as winning bids that in combination are willing to pay the most. The optimal selection of winning bids can be computed efficiently, which allows us to repeat this procedure at a high frequency.

Theorem 4.5.1 *Given a set of n intervals and corresponding prices. We can compute the optimal selection of intervals such that no two selected intervals are intersecting and the sum of prices is maximal in $O(n \log n)$ time or $O(n)$ if the intervals are already sorted by their end.*

Proof. This problem is also known as the *Weighted Interval Scheduling Problem*. It can be solved in $O(n \log n)$ resp. $O(n)$ via a simple dynamic program [103, Chapter 6.1]. □

Let $WIS(B, p)$ denote the optimal Weighted Interval Schedule for intervals B and prices/weights p . If there are multiple such solutions, return the one with the most intervals (can be easily integrated into the dynamic program).

4.5.2. Price Estimation

A natural approach to pricing a winning bid is the Vickrey principle, with the winning bid paying the price of the second highest bid. Applying this to our case with overlapping intervals requires some care, as there is no such thing as a clear “second highest bid”; e.g., the competition may be based on a long interval that has been outbid by multiple smaller intervals. To get a fair pricing, we set every price initially to some small value (such as zero or one) and increase the price only if the interval is not among the winners and their offer is not yet reached. This process ends when none of the prices of the losing bids can be increased anymore because they reached their current (maximum) offer. If the price of a winning bid is not limited by its offer, it would also have the same price even if its offer had been higher. This achieves the Vickrey principle, with the price mainly determined by the competition and not by the own offer. The algorithm is described in Algorithm 1, an example is given in Figure 4.3.

There are multiple options to increase the price, e.g., adding a constant value, a percentage of the current price, or a percentage of the maximum offer. Each of these solutions results in different runtimes.

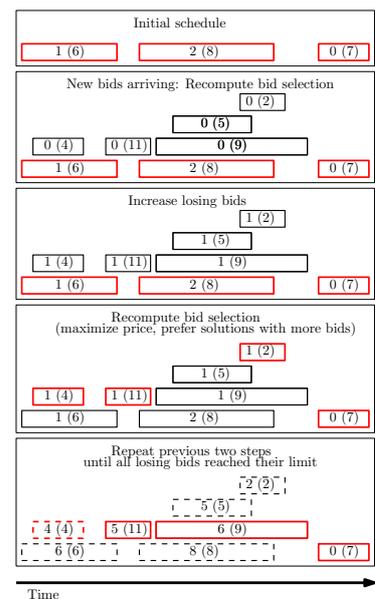


Figure 4.3.: Example of the auctioneer's algorithm. Every block is a bid for a time interval. It contains the current price and in brackets the limit. Red bids are the current bid selection. Dashed bids have reached their limit.

Algorithm 1: Auctioneer's Algorithm.

Data: A set B of bids with begin, end, and offer.
 A map $\text{price} : B \rightarrow \mathbb{R}_0^+$ with (initial) prices.
 A price increase function $\text{inc} : \mathbb{R}_0^+ \rightarrow \mathbb{R}_0^+$
Result: A subset $W \subseteq B$ of winning bids
 and a map of prices $\text{price} : W \rightarrow \mathbb{R}_0^+$

```

1  $W = \text{WIS}(B, \text{price});$ 
2 while  $\exists b \in B \setminus W : \text{price}[b] < b.\text{offer}$  do
3   for  $b \in B \setminus W$  do
4      $\text{price}[b] = \min(\text{inc}(\text{price}[b]), b.\text{offer});$ 
5    $W = \text{WIS}(B, \text{price});$ 
6 return  $W, \text{price};$ 

```

Theorem 4.5.2 Algorithm 1 with n bids and the highest offer being h terminates after at most

- ▶ $O(n^2 * h)$ steps for constant increments, e.g. +1.
- ▶ $O(n^2 * \log h)$ steps for proportional increments, e.g. +20% (initial price has to be positive).
- ▶ $O(n^2)$ steps for e.g., +0.05 * h .

Proof. In every step, at least one price is increased and every bid can only be increased $O(h)/O(\log h)/O(1)$ times. Sorting the bids for the end takes $O(n \log n)$ and reevaluating W takes $O(n)$. \square

Lemma 4.5.3 For a set of bids B , Algorithm 1 returns the same winning bids as $\text{WIS}(B, b \rightarrow b.\text{offer})$ if the solution is unique.

Proof. Further increments of the prices in Algorithm 1 do not change the winning bids, because all losing bids cannot increase their offer to become more attractive. The solution is thus identical to $\text{WIS}(B, b \rightarrow b.\text{offer})$. \square

In practice one can achieve a significant improvement by estimating the winning and losing bids first (because they do not have to pay anyways). Then we directly set the losing bids to their maximum price and prevent many alternating increments.

Theorem 4.5.4 If we set the price of the losing bids directly to their maximum and assuming the bids are sorted in advance, Algorithm 1 with n bids and the highest offer being h terminates after at most

- ▶ $O(n * h)$ steps for constant increments, e.g. +1.
- ▶ $O(n * \log h)$ steps for proportional increments, e.g. +20% (initial price has to be positive).
- ▶ $O(n)$ steps for e.g., +0.05 * h ,

Proof. Let W' be the winning bids, determined by $\text{WIS}(B, b \rightarrow b.\text{offer})$. In every step, $W \cap W'$ never decreases. Let w' be the last bid to be added to $W \cap W'$. It is incremented in every round, because it is losing in every

round except the last one. This is only possible $O(n)/O(\log n)/O(1)$ times. \square

By using a binary search to find the increment on which the solution changes can improve the runtime further. However, in both cases the prices can drastically differ to the original version.

4.6. Bidding Strategy

Every satellite can have its own strategy that best fits its purpose. A simple strategy can be to consider all feasible intervals on all ground stations and estimate their gain by using the value of the currently available (and not already scheduled) data and the prices of the intersecting highest bids. Then one bids on the interval with the highest expected gain and marks this interval and the corresponding data as scheduled (if this bid loses, this is undone). The maximum offer should not be the estimated value but only some percentage above the currently estimated price because at some price, another interval has a higher gain. This is repeated until there are no more intervals with positive gain on any ground station. The relative value of later bids decreases as the “good data” is already used for the previous bids. In order to discretize the set of intervals, we only consider those that begin at event points, such as the beginning of other highest bids, as well as periodical event points (e.g., every 5 minutes).

Future Work 4.1.

Can we refine this basic approach by one of the following methods?

- ▶ Use a prediction for future data value gain until the beginning of the considered interval.
- ▶ Consider the success probabilities of the previous bids for your value estimations and possibly even bid for conflicting intervals.
- ▶ Use the “second best opportunity” to estimate the maximum one would pay for an interval before switching to the second best. As intervals have different lengths, this is not trivial.
- ▶ Considering that intervals farther in the future may be subject to higher price increases.
- ▶ Multiple lower bids can outbid a higher bid. However, these bids are usually from different (not cooperating) satellites, so it is reasonable to occasionally try bids that do not seem to have a chance.

4.7. Experimental Evaluation of Schedule Quality

To validate the schedule quality of our method, we have evaluated four different satellite constellations, as shown in Table 4.1. They contain the Galileo constellation, a slightly enlarged constellation version of the Dove constellation of Planet, and one modeled after the planned OneWeb constellation. The Galileo constellation is in medium earth orbit and

Table 4.1.: Satellite constellations used for experimental evaluation.

Constellation	1	2	3	4
Inspiration	Galileo	Artificial	Dove	OneWeb
Number of S/C	39	40	400	1080
Orbit region	MEO	LEO	LEO	LEO
Inclination/deg	56	85	97	87.9
Eccentricity/-	0.001	0.001	0.001	0.001
Number of ground stations	2	6	5	6

is significantly slower, i.e., passes and pauses are much longer than for the LEO constellations. Each of the four cases was simulated for a schedule of 36 days to compute realistic contact windows. The satellites continuously generate data whose size and value is determined by a uniform probability distribution. Details on the simulation are described by Schaus et al. [17].

4.7.1. Greedy Algorithm

To get a comparison for the schedule quality, we also implemented a greedy algorithm. It schedules for a time interval (e.g., 1-3 hours) all intervals by iteratively selecting the contact with the highest (relative) download value until no more contacts can be scheduled. We use one variant with the highest absolute value and one variant with the highest relative value (value/length). On tie, the shorter contact is chosen such that conflicts are minimized. This is fairly similar to how a human approaches this problem.

4.7.2. Experimental Results

In Figure 4.4 one can see that for the second constellation, almost all data can be downloaded, even with a greedy approach. For the third and fourth constellation, our auction-based approach still achieves a relatively high rate, while the greedy approach drops considerably. The reason is that it is no longer possible to download all data, so an algorithm has to make the best choices; this also implies an increasing difference between value rate and data rate. An excerpt of a schedule is shown in Figure 4.6. The results for Galileo can be explained by the long contact-less intervals that keep a lot of the data out of reach.

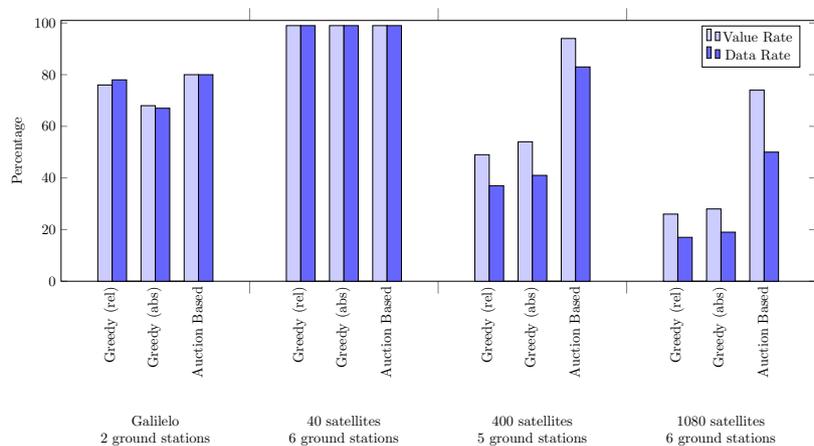


Figure 4.4.: Data and value rate for different constellations over 36 days. For the first two experiments with few satellites, the performance of the greedy approaches and our auction-based approach are relatively similar. However, the performance of the greedy approaches drops significantly for the two larger experiments while our auction-based approach still achieves relatively good rates.

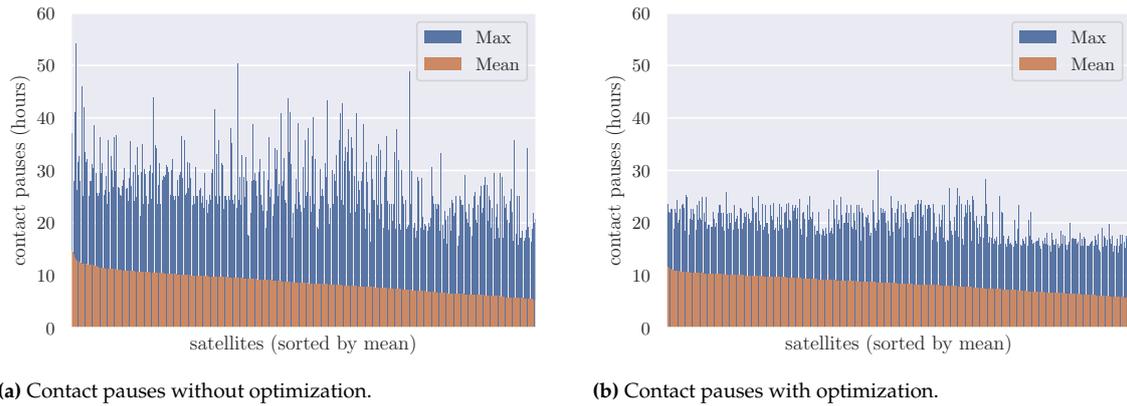


Figure 4.5.: If you want to keep the contact pauses of a satellite short, you can increase the bids with the length of the current contact pause in order to easily minimize those. The optimization cuts the maximum pause almost in half for the Dove-like constellation while barely reducing the data and value rate.

4.7.3. Contact Pauses

Besides the value of the data, one is also interested in minimizing the maximal pauses between two contacts, i.e., the continuous time without contact. Both greedy and our auction-based algorithm can have relatively long pauses without further adjustments. However, the bidder strategy of the satellites can easily be adapted to keep these pauses small. If the last scheduled contact of a satellite is too old, the estimated value is increased based on the time distance such that it continuously increases. The satellite will now give continuously increasing bids, until it wins a bid after which this “boost” is reset. The result are much shorter maximum contact pauses for the satellites (see Figure 4.5), while the downloaded value is only slightly worse. This shows the flexibility of our approach.

4.8. Conclusion

In this chapter, we have seen an auction-based method for optimizing the download schedules for large-scale constellations of spacecraft. Our simulation results on realistic constellations demonstrate the power of

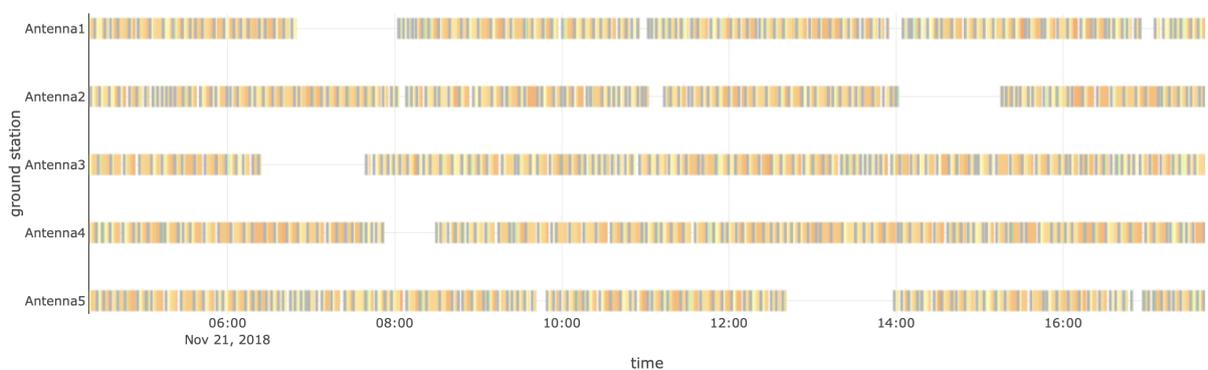


Figure 4.6.: An excerpt of the schedule for the Dove-like instance. The gray strips are adjustments, the colored strips visualize the value of downloaded data packages. Note that even for 400 satellites, there are still pauses in which no satellite is in range of the ground station. During the other times, the schedule is packed tightly. The larger OneWeb-like instance does not have any pauses.

this approach even when using a simple bidding strategy. Ideas for designing more advanced bidding strategies that potentially further improve the performance of this approach are listed in Section 4.6. Future developments can be expected from inter-satellite communications, as considered in the previous two chapters, which will give rise to even more complex perspectives of collecting and aggregating data, as well as more flexibility than strategies involving fixed ground stations.

Part II.

**PARTIAL COVERAGE PATH
PLANNING**

Coverage path planning is a fundamental task in mobile robotics, manufacturing, and other areas. It allows us not only to optimize industrial operations like CNC-milling, UAV trajectories, and the tilling and mowing of fields, but can also be employed in personal contexts, e.g., in the form of a robot vacuum cleaner, which you perhaps already own and use to clean your living room. There exist various different techniques for coverage path planning, some more advanced than others. Some robot vacuum cleaners execute their task of cleaning using random patterns, while others actually map the owner's apartment (with varying success) and compute smarter movement patterns (with varying success).

The theoretical work in the current literature mostly considered highly idealized robot models and environments. By partitioning the area into robot-sized tiles, we can easily transform the problem into the TRAVELING SALESMAN PROBLEM; this may be NP-hard, but received so much attention from algorithm engineers that even large instances can usually be solved to near-optimality. Unfortunately, this completely ignores the dynamic of the robot and only minimizes the traveled distance. While this is sufficient for large-scale routing problems like car navigation where the necessary acceleration after turns is negligible for the overall tour, this can be critical for turn-heavy coverage tours, e.g., those used by robot vacuum cleaners or harvester. Minimizing the distance for coverage trajectories can even introduce many expensive turns in order to prevent redundant coverages and minimize the traveled distance.

In Chapter 5, we engineer the approximation algorithm for tours with turn costs in grid graphs, introduced in [18, 27], to solve instances with up to 300 000 vertices. The computational evaluation shows that this algorithm yields nearly optimal solutions and performs better than the only other known approximation algorithm. Additionally, the algorithm supports penalty and subset variants in which not all vertices need to be visited.

As the world is not a grid graph, Chapter 6 generalizes this approach to finding partial coverage paths in complex polygonal environments. Partial coverage refers to the ability to focus on covering areas deemed more important, given by a set of weighted polygons. Additionally, it is also possible to vary the costs in specific areas. This is achieved by fitting a grid or mesh onto the area and converting it to a discrete problem. The corresponding study can also be highly relevant for other grid-based covering algorithms.

This part furthers the research started in my master's thesis by transforming its theoretical results into something practically applicable.

Engineering an Approximation Algorithm*

5.

This chapter engineers an approximation algorithm for tours and cycle covers with turn costs in grid graphs such that instances with over 300 000 points can be solved. Theoretical insights are used to transform the instances into an equivalent but smaller problem. The evaluation afterward shows that the algorithm is often close to the optimum and has on average a 70 % smaller optimality gap than the previous algorithm. Additionally, the extension to penalty coverage is considered and evaluated.

- 5.1 Introduction 95
 - 5.1.1 Related Work 96
 - 5.1.2 Overview 98
- 5.2 Preliminaries 99
- 5.3 Efficient Implementation 99
 - 5.3.1 Atomic Strip Covers . . . 100
 - 5.3.2 Matching 102
 - 5.3.3 Partial Coverage 102
 - 5.3.4 Tours 103
 - 5.3.5 Other Grids 103
- 5.4 Evaluation 104
- 5.5 Conclusion 107

5.1. Introduction

The TRAVELING SALESMAN PROBLEM (TSP) is one of the classic tasks of combinatorial optimization. Easy to describe but NP-hard to solve, it has given rise to a large body of research focusing on theoretical aspects such as complexity and approximation. On the practical side, the TSP has also stimulated significant work in algorithm engineering. Ever since Dantzig, Fulkerson and Johnson [58] in 1954 presented the provably optimal solution of a 49-city instance based on an (integer) linear-programming (IP/LP) approach, IP/LP methods have been used on a wide spectrum of other optimization problems. With a variety of additional techniques, the frontiers of TSP instance sizes for which provably optimal solutions can be computed have been pushed all the way to 85,900 cities; see [104] for a comprehensive overview.

Research on TSP has also served as the blueprint for a wide range of other real-world problems, many of them motivated by generalizations or modifications, such as lawn mowing and milling, where visiting a discrete set of points is replaced by covering a geometric region with a tool or sensor. Other variants arise from modified objective functions or additional constraints, such as the total turn of a tour instead of its length.

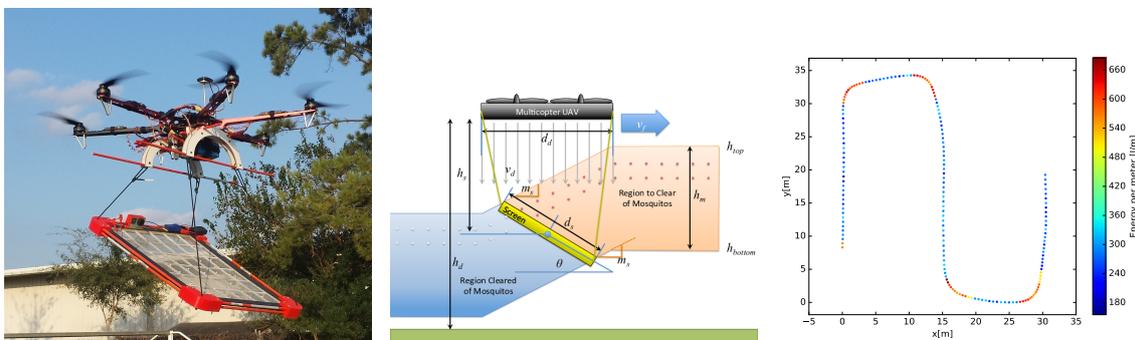


Figure 5.1: (Left) A drone equipped with an electrical grid for killing mosquitoes. (Middle) Physical aspects of the flying drone. (Right) Making turns is expensive. See our related video at <https://www.youtube.com/watch?v=SFyOMDgdNao> for details, and [20] for an accompanying abstract.

* The content of this chapter was presented at ALENEX 2019 [5].

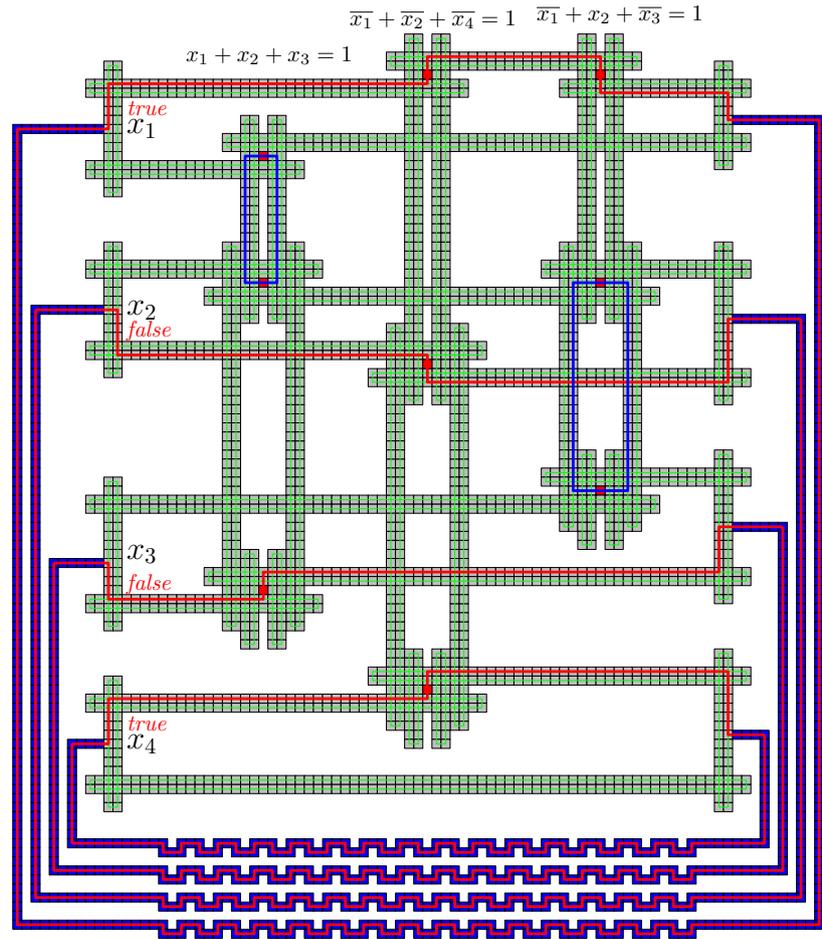


Figure 5.2.: Example of the hardness-construction for the ONE-IN-THREE-SAT-clause $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$ [18]. The size and complexity was strongly reduced since its first appearance in the master’s thesis [27].

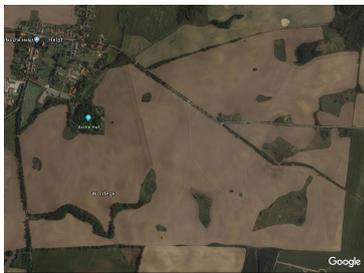


Figure 5.3.: Satellite image of a real-world instance arising from mechanized agriculture. Notice the tracks resulting from the use of large-scale agricultural machinery, and the uneven crop yield, motivating subset and penalty versions of the problem. (Image: Google, GeoBasis-DE/BKG ©2009)

In this chapter we present how to engineer a theoretical approximation algorithm for computing covering tours and cycle covers with turn cost, which are of practical significance in areas such as pest control (see Figure 5.1) and precision farming (see Figure 5.3). (See the related video [20] <https://www.youtube.com/watch?v=SFy0MDgdNao> for an animated illustration in the context of fighting mosquitoes.) This includes variants such as the subset and the penalty versions of the problem, in which the objective is to cover appropriate subsets of the given region. True to the spirit of algorithm engineering, the work presented in this chapter refines and extends the previous theoretical work [18] that discusses complexity and approximation. This previous work provides a constant factor approximation algorithm for full, subset, and penalty coverage with turn costs in grid graphs which we engineer in this chapter. It also proves the NP-hardness of the cycle cover relaxation of the problem whose complexity has been open for many years as *Problem 53* in *The Open Problems Project* [28]. An example of the construction is shown in Figure 5.2.

5.1.1. Related Work

The problem of minimizing the necessary distance for covering a given region by a moving tool is known as the LAWNMOWER PROBLEM; if the tool is required to stay within the region, we are dealing with the MILLING

PROBLEM. See Arkin et al. [105] for a theoretical study, providing approximation algorithms for both variants. They give a 2.5-approximation for minimum length milling in orthogonal (not necessarily integral) polygons with a unit square cutter. For simple orthogonal polygons, an $11/5$ -approximation is given and, in case we can reduce the problem to finding a covering tour in a grid graph, a $6/5$ -approximation algorithm is given (which improves a previous result of Ntafos [106] of a $4/3$ -approximation algorithm). For the mowing variant, a $3 + \epsilon$ -approximation is provided that internally uses a PTAS for the Euclidean TSP.

For covering a discrete set of points in the \mathbb{R}^2 plane for which only the turning angles are measured, the problem is called the ANGULAR METRIC TSP. The cost function changes for this problem from a relatively straightforward sum of edge weights to more involved combinations of edges at the vertices, making approximation more challenging. Aggarwal et al. [34] provide an $O(\log n)$ approximation algorithm for cycle covers and tours that works even for distance costs and higher dimensions. Fekete and Woeginger [36] consider the problem of connecting a point set with a tour for which the angles between the two successive edges are constrained. Finding a curvature-constrained shortest *path* with obstacles has been shown to be NP-hard by Lazard et al. [107]. Without obstacles, the problem is known as the DUBINS PATH [108] that can be computed efficiently. For different types of obstacles, Boissonnat and Lazard [109], Agarwal et al. [110] and Agarwal and Wang [111] provide polynomial-time algorithms or $1 + \epsilon$ approximation algorithms, respectively. Takei et al. [112] consider the solution of the problem from a practical perspective. The DUBINS TRAVELING SALESMAN PROBLEM is considered by Le Ny et al. [113].

For covering a geometric region in the presence of turn cost, Arkin et al. [32, 33] provide a first approximation algorithm for tours and cycle covers in grid graphs with turn cost and show hardness of the tour variant. Most closely related to this chapter is the theoretical work of Fekete and Krupke [18] that provides important theoretical progress, resolving *Problem 53* in *The Open Problems Project* [28] by proving that finding a cycle cover of minimum turn cost is NP-hard, even in the restricted case of grid graphs. As a consequence, all relevant problem variants are NP-hard. They also proved that finding a subset cycle cover of minimum turn cost is NP-hard, even in the restricted case of *thin* grid graphs, in which no induced 2×2 subgraph exists. This differs from the case of full coverage in thin grid graphs, which is known to be polynomially solvable [33]. They also provided a general IP/LP-based technique for obtaining constant-factor approximations for all problem variants; some details are described in Section 5.3. This approach includes the first approximation algorithms for subset cycle covers and tours, as well as for penalty cycle covers and tours; these are also valid for travel costs that are linear combinations of turn and distance costs.

Algorithm engineering for covering tours and cycle covers with turn cost has been more limited. Maurer [114] proves that cycle *partition* with turn cost in grid graphs can be solved in polynomial time. He also performed integer programming experiments on cycle cover and cycle partition. De Assis and de Souza [115] considered integer programming for tours but were only able to solve small instances with up to 76 vertices. Fekete and

Krupke [5] were able to solve instances with more than 1000 vertices to optimality.

The generalization of the ANGULAR METRIC TSP to generic graphs is called the QUADRATIC TRAVELING SALESMAN PROBLEM where the cost of an edge can depend on the preceding edge. The solvable instances even for the geometric angular metric variant are usually very small (below 100 points), see, e.g., [116], [117], [118].

Covering a polygonal area by a tour is an important practical problem known as *Coverage Path Planning* and, hence, a considerable amount of work can be found. Two common techniques are putting a grid on the area as we do, see, e.g., Zelinsky et al. [119] or Gabriely and Rimon [120], or partitioning the polygon into simple geometric areas (e.g. trapezoidal map) and use simple patterns to cover these areas, see, e.g., Huang [121]. Optimizing cutter paths is considered by Yao et al. [122] who also mention the increase of costs due to turns. Planning tractor tours for crop harvesting operations including turn penalties is analyzed by Ali et al. [123]. Ahmadzadeh et al. [124] and Agarwal et al. [125] look at covering tours of UAVs for area observation with restrictions on the turn radii. We will revisit the practical aspects of this problem again in the next chapter and provide more related practical work in Section 6.3 (Related Work) on page 112.

An novel aspect of our approximation algorithm is its ability for partial coverage, i.e., having optional fields that only induce a penalty on neglecting. This is strongly related to the Penalty TSP, except for the turn costs. An overview of such problems and their algorithms, without turn costs, is given by Ausiello et al. [126]. Turn costs can partially modelled by a GENERALIZED TSP (GTSP) in which we have to visit sets instead of individual vertices. We can model different turns through a vertex by different vertices in the GTSP's sets. Helsgaun [127] solved instances with up to 17 180 sets and 85 900 vertices to near optimality.

5.1.2. Overview

In this chapter we bridge the gap between theory and practice of finding covering tours and cycle covers with turn cost, and show how to apply algorithm engineering techniques in combination with refined modeling in order to compute near-optimal solutions for large instances.

We provide the following main results.

- ▶ We describe an efficient implementation of an approximation technique for full/subset/penalty coverage with cycle covers or tours.
- ▶ We present a comprehensive computational study of instances with up to 300 000 pixels, for which we give solutions that are typically within a few percent of the computed lower bounds.
- ▶ We also provide a practical comparison of our approximation algorithm with the approximation technique of Arkin et al. [105] that dates back to 2001; we show that our new LP/IP-based approximation method closes 70 % of the remaining optimality gap to the lower bound for a wide range of benchmark instances.

5.2. Preliminaries

Given a grid graph $G = (P, E)$, where P are unit-sized squares on the integral grid, also called *pixels*, and two pixels are joined by an edge $e \in E$ iff they are adjacent. In addition, we may be given a subset $S \subseteq P$ of pixels or a penalty function $\rho : P \rightarrow \mathbb{Q}_0^+$. We consider tours and cycle covers in grid graphs with three variations: *Full coverage*, where every pixel has to be covered, *subset coverage*, where at least a specific subset S has to be covered, and *penalty coverage*, where every uncovered pixel $p \in P$ involves a penalty $\rho(p)$. The objective function is an arbitrary but fixed non-negative linear combination of the number of pixel transitions and turns. For the penalty variant, the objective function also contains the sum of penalties for uncovered pixel. The number of turns is measured in 90° turns, which we also call *simple turns*. The cost of a u-turn (a turn of 180°) corresponds to two simple turns. A *cycle* is a closed sequence of at least two adjacent pixels. A pixel is covered by a cycle if it is in its pixel sequence. A *cycle cover* is a set of cycles that together cover all pixels, while a *tour* is a cycle cover that consists of a single cycle. See Figure 5.4 for an example.

To improve the performance (especially for the matching algorithm), we only use integral weights and penalties. It is easily possible to adapt the implementation to floating point values. However, one can also approximate floating point values by scaling. In this case, one has only to be careful not to create overflows in any part of the algorithm.

5.3. Efficient Implementation

Fekete and Krupke describe in [18] an approximation technique that yields constant-factor approximations for full/subset/penalty cycle covers and tours in grid graphs (and even more generic geometric instances). The purpose of that proof was to establish a constant factor for the worst-case performance, not practical efficiency. As a consequence, the theoretical algorithm would struggle (especially regarding memory consumption) with instances larger than 10 000 pixels, even when exploiting duality properties to reduce the size of the auxiliary problems. In this section we give a number of additional algorithm engineering techniques to turn this worst-case performance into a significantly more efficient implementation, based on exploiting more refined properties. The result is an implementation that is able to solve instances with hundreds of thousands of pixels. The corresponding experimental evaluation is given in Section 5.4. For easier description, we initially focus on full coverage cycle covers. The necessary adaptations for the other variants are sketched in the end.

A fundamental part of the approximation technique are *atomic strips*, which allow us to push the turn cost into the edge weights. Similar to vertices in a grid graph, they encode *positions*; in addition each atomic strip also encodes an *orientation* by having two entries/exits in opposite directions. Hence, an atomic strip can be interpreted as a strip or line of zero (or infinitely small) length. For each pixel, we have a horizontal and a vertical atomic strip, of which at least one has to be in the solution to cover the pixel. Thus, an *Atomic Strip Cover* (ASC) is a selection of

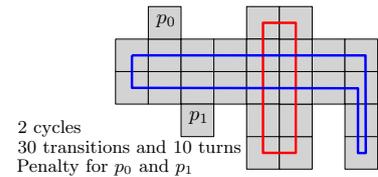
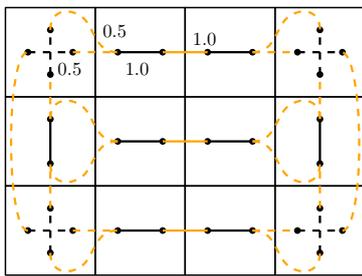
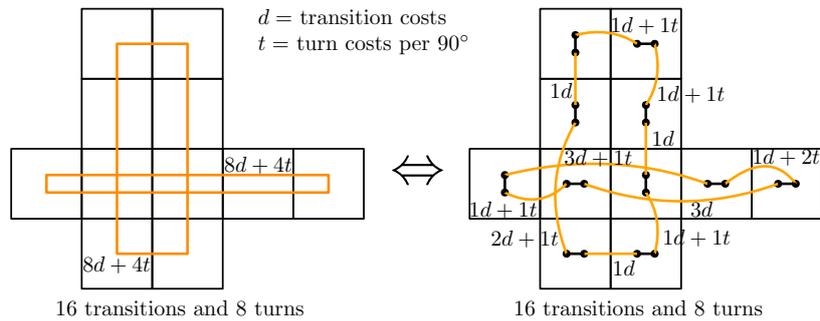
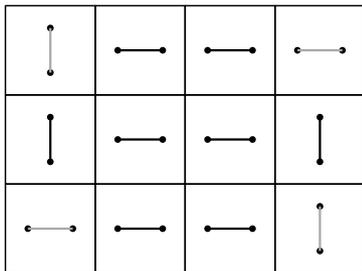


Figure 5.4. Example of a penalty cycle cover whose cost is a linear combination of the number of transitions and 90° turns and the penalties for not covering p_1 and p_2 .

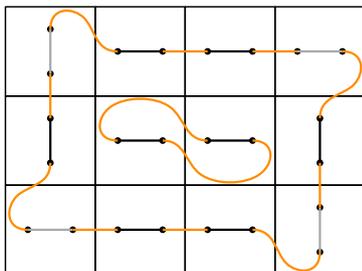
Figure 5.5: Finding the cheapest cycle cover and finding the ASC with the lowest perfect matching are equivalent problems. If we select the atomic strip that has either the correct entry or exit orientation, the cost of the matching edges equals the cost of the cycle. If the wrong atomic strip is selected, the cost can increase by two turns. The weight of the matching edges can be computed efficiently (in our efficient implementation, this is actually not necessary). Covering a pixel multiple times is not a problem, because the matching edges can skip pixels (for recreating the cycle cover, one just has to keep track of which ones).



1. Get fractional solution



2. Extract Atomic Strip Cover



3. Match atomic strips

Figure 5.6.: Example of the approximation algorithm for a simple full cycle cover in a grid graph. First a fractional solution for the atomic strip cover matching formulation is computed via linear programming. Strips and edges with value 0 are omitted, while dashed ones have a value of 0.5. Then the dominant (i.e., highest valued) atomic strips of this solution are selected. The gray atomic strips are ambiguous, i.e., we could have also chosen the other one. Finally, a minimum weight perfect matching on the ends of the atomic strips is computed. Recall that the atomic strips do not have any length (but only an orientation) so the curves in the corner are actually simple 90° turns.

exactly one *atomic strip* per pixel. A cycle cover is obtained by computing a minimum weight perfect matching on the endpoints of the selected atomic strips. The weight of a matching edge equals the minimum cost of transiting between the two endpoints including the turns at the ends. It is straightforward to prove that the minimal matching over all possible ASCs corresponds to an optimal solution, see Figure 5.5 and [18] for more theoretical details.

In the original algorithm we use an integer program (IP) that combines finding this ASC with its corresponding matching. This IP is then solved fractionally; for each pixel, the strip with the higher fractional value for an ASC is selected. We were able to show that the matching of this ASC yields a solution at most 4 times higher than the optimal solution by using polyhedral arguments. See Figure 5.6 for an illustration.

We first show how to implement the ASC and matching parts of the approximation algorithm for (full) cycle cover. Afterwards we sketch how to obtain algorithms for subset and penalty coverage, as well as the tour variants. An extension to other grids is also possible.

5.3.1. Atomic Strip Covers

Formulating the problem of finding the ASC with the best matching as an integer program requires $O(|P|)$ Boolean variables for the atomic strips and $O(|P|^2)$ Boolean variables for the matching edges. In addition, the weight of the matching edges has to be computed, which can be very time-consuming for large grid graphs.

There are multiple possibilities for formulating the cycle cover problem as an integer program. Almost all of these formulations are potential replacements for the linear relaxation of the original integer program, because the solutions can easily be transformed and the proof in [5] only requires a fractional solution that is a lower bound on the optimum.

Lemma 5.3.1 ([18]) *A fractional solution for Atomic Strip Cover and matching (see Figure 5.6) can be transformed into an integral solution of at most four times the objective value.*

We extract the Atomic Strip Cover from the fractional solution of the following IP, which we selected based on prior experiments. We use the non-negative variables $x_{ijk} = x_{kji} \in \mathbb{N}_0$ for pixel $p_j \in P$ and adjacent

pixels $p_i, p_k \in N(p_j)$ that state how often the transition $p_i - p_j - p_k$ or $p_k - p_j - p_i$ is used, see Figure 5.7.

Let $\text{cost}_j(i, k) \in \mathbb{Q}_0^+$ map the cost of this transition. We minimize the overall coverage costs; Eq. (5.2) enforces a pixel to be covered and Eq. (5.3) enforces the transitions between two adjacent pixels to match.

$$\min \sum_{p_j \in P} \sum_{p_i, p_k \in N(p_j)} \text{cost}_j(i, k) \cdot x_{ijk} \tag{5.1}$$

$$\text{s.t.} \quad \sum_{p_i, p_k \in N(p_j)} x_{ijk} \geq 1 \quad \forall p_j \in P \tag{5.2}$$

$$2 \cdot x_{jij} + \sum_{p_k \in N(p_i), p_k \neq p_j} x_{jik} = \tag{5.3}$$

$$2 \cdot x_{iji} + \sum_{p_k \in N(p_j), p_k \neq p_i} x_{ijk} \quad \forall \{p_i, p_j\} \in E$$

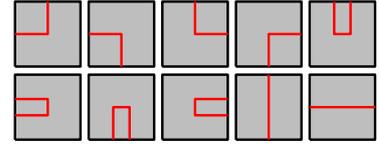


Figure 5.7.: There are 10 ways of covering a pixel; each one corresponds to a variable of the IP.

Obtaining a fractional Atomic Strip Cover from the fractional solution of this IP is done as follows. If a coverage variable represents a straight transition, add its value to the equally oriented strip, otherwise distribute the value equally between both, because both are valid selections. Our approximation algorithm selects the dominant strip, i.e., the strip with the highest value in the fractional solution. See Figure 5.8 for an illustration. This Atomic Strip Cover equals the selection of an equal weighted fractional Atomic Strip Cover and matching as in the original algorithm in [18] and is hence a feasible replacement.

Lemma 5.3.2 *A fractional solution of the IP (5.1)-(5.3) can be converted into a fractional solution (of equal objective value) of the original IP, i.e., a fractional Atomic Strip Cover with a corresponding fractional matching.*

Proof. It is easy to see that for an integral cycle we can easily extract matching atomic strips and corresponding connecting edges of equal cost. For transforming a fractional solution, simply multiply the solution by some value z such that all cycles become integral. Do the transformation independently, i.e., do not care for double coverages, for every integral cycle and afterwards divide the solution again by z . Remove the superfluous coverages by connecting matching edges at both ends of an overused atomic strip directly which does not increase the cost but decreases the usage of the atomic strip. Note that the original integer program allowed loop edges in the fractional solution as they cannot appear in integral solutions and do not harm the proof. \square

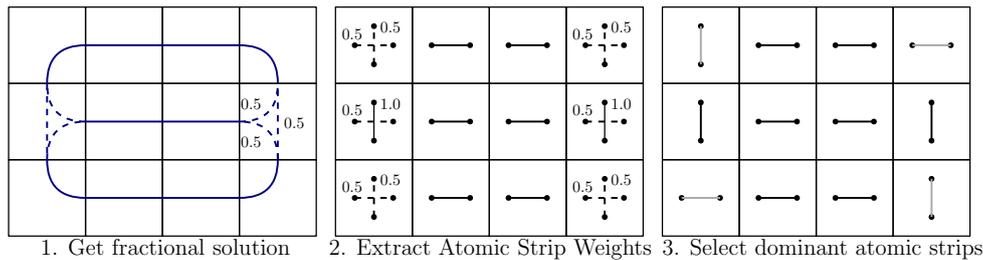
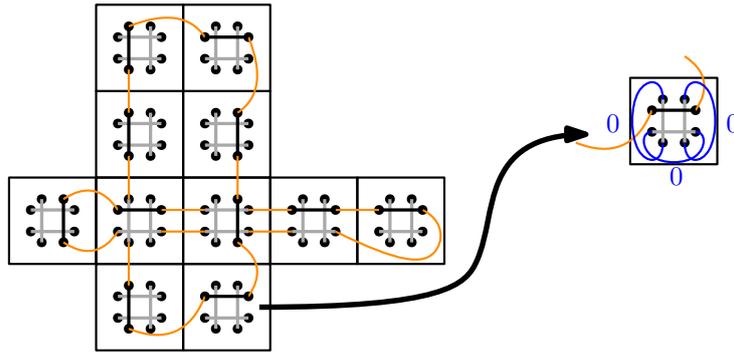


Figure 5.8.: Obtaining the Atomic Strip Cover from the IP (5.1)-(5.3). Note that the overcoverage of some pixel is not a problem, because we can reduce it by connecting two matching edges of the endpoints of the atomic strip, i.e., skip it without increasing the cost.

Figure 5.9.: If we add optional atomic strips (gray), we only need matching edges between atomic strips of adjacent pixel. Optional atomic strips can be created by adding a zero-cost edge between its endpoints that allow it to be neutralized if not needed. We need at most three such optional atomic strips per pixel.



Future Work 5.1.

Can we improve the LP-relaxation by some additional constraints? If we reduce the integrality-gap, we directly improve the proved approximation factor.

5.3.2. Matching

We now have an atomic strip cover, i.e., an atomic strip per pixel, that we now need to connect to a cycle cover via a matching on the end points. A naïve matching of the strips results in a quadratic size matching graph, where each edge represents a connection that has to be computed and stored. Even when utilizing the dual of the primal-dual matching algorithm to exclude edges, the memory and time consumption showed to be too high for larger instances ($> 10\,000$ pixels). However, one can use the simple structure of a grid graph to solve this problem much more efficiently: The matching in Figure 5.5 can also be expressed as a matching that only uses short edges between two adjacent pixels if we are allowed to add additional strips, see Figure 5.9. We can add such optional strips by having a zero-cost edge between its two endpoints, such that it disappears without cost if it is not needed. Recall that we match the end points of the atomic strips; thus, using the zero-cost edge effectively removes the corresponding atomic strip. Further, we can limit the maximally needed strips to two horizontal and two vertical strips due to the following lemma.

Lemma 5.3.3 (Arkin et al. [33]) *If a cycle cover covers a pixel more than four times or is passed straight more than two times in the same orientation, the length of the cycle cover and the coverage of the pixel can be reduced by local optimization without increasing the turns.*

The strip of the Atomic Strip Cover does not get such a removal edge, because it has to be used by the proper cycles, so we are left with 8^2 edges per two adjacent pixels and three zero-cost edges per pixel.

5.3.3. Partial Coverage

Subset coverage can be easily expressed as penalty coverage by making the penalty extremely high for mandatory pixels and zero for optional pixels. So we can limit ourselves to the penalty coverage variant. The idea

for penalty coverage is to introduce an artificial cycle for each pixel with the cost of the penalty that covers exactly this pixel. For this we can add an edge to each atomic strip that connects the both endpoints. The weight of such an edge is the penalty for not covering the corresponding cycle. When using IP (5.1)-(5.3) we only need to add an additional Boolean variable for each pixel that expresses using the corresponding penalty cycle. This variable needs to be added to the objective function and the coverage constraint. Whenever the penalty is too high to be a reasonable option for a pixel, this step can also be skipped and the pixel can be treated like for full-coverage. See [18] for more details on this idea.

5.3.4. Tours

For full coverage one can always connect two adjacent cycles with at most two additional turns and two additional transitions as shown by Arkin et al. [33]. They also showed that if a pixel is covered more than four times by a tour, we can do a local optimization. Because every cycle has at least four turns and two transitions on its own, greedily connecting adjacent cycles provides us an approximation factor of 6.

For subset coverage we do not necessarily have adjacent cycles but one can build a graph where every cycle is a vertex and every edge is the cheapest connection between these two cycles. In [18] we show that connecting the cycles to a tour via a minimum spanning tree in this graph (see Figure 5.10) provides us an approximation factor of 10.

For penalty coverage we also have to consider that we could decide not to cover pixels instead of connecting the cycles. In [18] we show that by using a 2-approximation for the Prize-Collecting Steiner Tree instead of a Minimum Spanning Tree, one can obtain a 12-approximation.

5.3.5. Other Grids

The approximation technique including the just presented techniques for runtime improvement can also be applied to more generic grid graphs such as triangular or three-dimensional grids. For triangular or three-dimensional grids we would need three atomic strips per vertex. In general one has to make sure that for every possible coverage possibility of a vertex/pixel either incoming or the outgoing orientation matches the endpoint of an atomic strip. Of course this may need more optional atomic strips for the matching technique which can become problematic already for triangular grids. Note that while we are using uniform distance costs here, the algorithm theoretically allows the usage of heterogeneous edge lengths. Also the turning angles would allow some heterogeneity and hence one could for example also compute solutions for 'warped' grid graphs. Obtaining 'well deformed' grid graphs for natural instances, e.g., fields for harvesting, instead of just putting a uniform grid over it is a potential next step to improve the practicality of this approach and is part of Chapter 6.

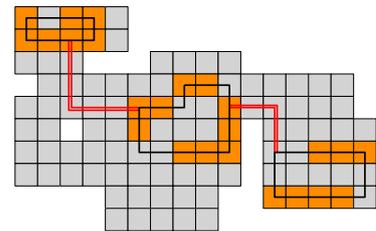


Figure 5.10.: Connecting subset cycles (cycles in black, subset pixel in orange) by a minimum spanning tree (red edges) on the components/cycles.

5.4. Evaluation

We implemented and tuned our approximation technique for full, subset, and penalty cycle covers and tours in grid graphs. We used CPLEX 12.7.1.0 and the minimum-weight perfect matching implementation of Kolmogorov [128]. For connecting the penalty cycle cover to a tour, we used a Prize-Collection Steiner Tree implementation (proved 2-approximation) of Hedge, Indyk, and Schmidt [129]. Furthermore, we implemented the approximation algorithm of Arkin et al. as a comparison. The (C++)-code, test instance examples, and further material can be found on <https://github.com/d-krupke/turncost>.

We used two different generators for creating random test instances that are motivated by floor plans. The first one (Type I) uses a random orthogonal polygon with 20 to 200 vertices and selects all points of a grid inside of it; see Figure 5.11 for an example. The second one (Type II) takes a union of random rectangles, with the maximum size of a rectangle bounded by $O(\sqrt{N})$, N being the desired size. We used two different parameter settings to obtain instances of different granularity, denoted by Type IIa and Type IIb; examples are shown in Figure 5.12 for Type IIa and Figure 5.13 for Type IIb. While the test instances for Type II have a uniform size distribution, for Type I there are more smaller than larger instances, because the amount of pixel is harder to control in the generation process.



Figure 5.11.: An example instance of Type I with 7684 pixels.

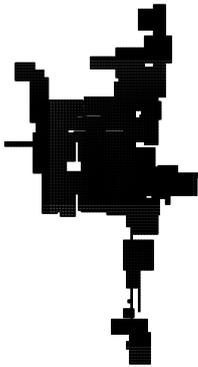


Figure 5.12.: An example instance of Type IIa with 10052 pixels.

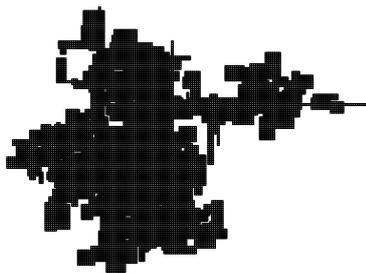


Figure 5.13.: An example instance of Type IIb with 10004 pixels.

The practical computations were performed on regular desktop computers equipped with an *Intel(R) Core(TM) i7-6700K CPU @ 4.00 GHz* and 64 GB of RAM. The used CPLEX version was 12.7.1.0. In all cases, the objective function models the travel cost arising from a linear combination of turn and distance cost, with a range of different coefficients, accounting for different relative cost of making turns.

For full cycle covers and tours, Figure 5.14 gives a comparison and ground truth of our approximation algorithm with the provably optimal objective values for instance sizes up to about 1000 pixels; it can be seen that the computed value is mostly within 1% to 2% resp. 2% to 5% of the optimum. Instances of Type IIb seem to be slightly better which can be explained by the rougher boundary that has more necessary turns increasing the objective.

The performance of our new approximation method (FK) and the one by Arkin et al. (ABD+) for larger instances is analyzed in Figure 5.15. We use the lower bound provided by the linear relaxation as a reference such that the y-axis is an upper bound on the relative optimality gap. A value of 5% means that the solution is on average at most 5% above the optimal objective. We can see that overall, FK closes about 70% of the gap left by ABD+. The performance of FK even improves for larger instances, especially for instances of Type I. This can be explained by the instance generation that tends to increase the area faster than the boundary complexity. Instances of Type IIb have the highest optimality gap which is slightly contradicting to the previous experiment. This can be explained by a weaker linear relaxation on these complex instances with many turns. Especially, $1/2$ -u-turns on already covered pixels allow to create paths instead of cycles in the linear relaxation. Tours show, as expected, to have a slightly higher (+2% to 3%) optimality gap but

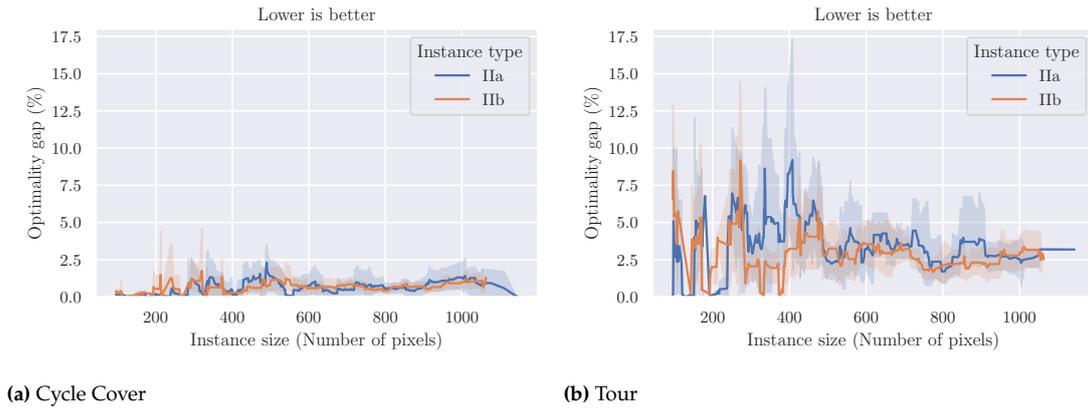


Figure 5.14.: Relative performance of the new approximation algorithm compared to the optimal solution, for instances of Type IIa and Type IIb. For cycle covers (left) it can be seen that the approximation algorithm mostly produces very small gaps for cycle cover; it often finds the optimum, most of the other times it is within less than 1% to 2% of the optimum. This indicates that for larger instances, a major part of the remaining gap may be due to the heuristic lower bound, implying that the quality of the found solution is even better. For tours (right), the performance is worse, due to heuristic greedy merging of cycles into a tour. The objective function is a linear combination of 50 times the number of simple turns, plus the number of pixel transitions. Only instances that could be solved within 15 min to optimality by an integer program [5] are considered (318 for cycle cover, 270 for tours).

overall it remains well below 10% for our algorithm. Only for small instances of Type I there are higher values. Note that the plots only show the mean and the 95%-confidence interval; there are some outliers in which our algorithm is around 50% above the lower bound.

The weight of the turn costs has an influence for instances of Type I and IIa, with a higher weight resulting in a higher gap. For instances of Type IIb, the turn costs seem to have no clear effect except that the variance is reduced. The results for ABD+ regarding the weight of the turn costs can be slightly confusing as it seems shuffled. This is because we take the weights into account for the matching phase of this algorithm. While this does not correspond to the original version, it can only improve the result. If we would not do this, the lines should be ordered as they would be based on the same solutions. The fractional solutions can differ, too, but should show similar results for FK if they would be the culprit.

A runtime comparison for our approximation algorithm for cycle cover is given in Figure 5.16. For some instances of Type II (in particular, for those of Type IIb with the “roughest” boundaries), the runtime is affected by the effort for solving the linear program and computing the matching (as shown in Figure 5.16). While a higher turn cost seems to affect the runtime negatively, this effect is low compared to the general deviation. Our optimized implementation of our new approximation method (FK) was frequently faster than our naïve implementation of the approximation algorithm by Arkin et al. (ABD+); however, this is an unfair comparison and an optimized implementation of the later should be visibly faster such that we only have an advantage regarding the solution quality.

Finally, results for the penalty variants of optimal cycle covers and tours for instances of Type IIa up to 50 000 pixels are shown in Figure 5.17. Despite the more involved objective function and solution space, the optimality gap compared to a fractional lower bound is typically about 20% and never more than 50%, even for the tour version. The higher the penalty, the better is the value. This can be explained by that it becomes closer to the original full-coverage version with a higher penalty as less

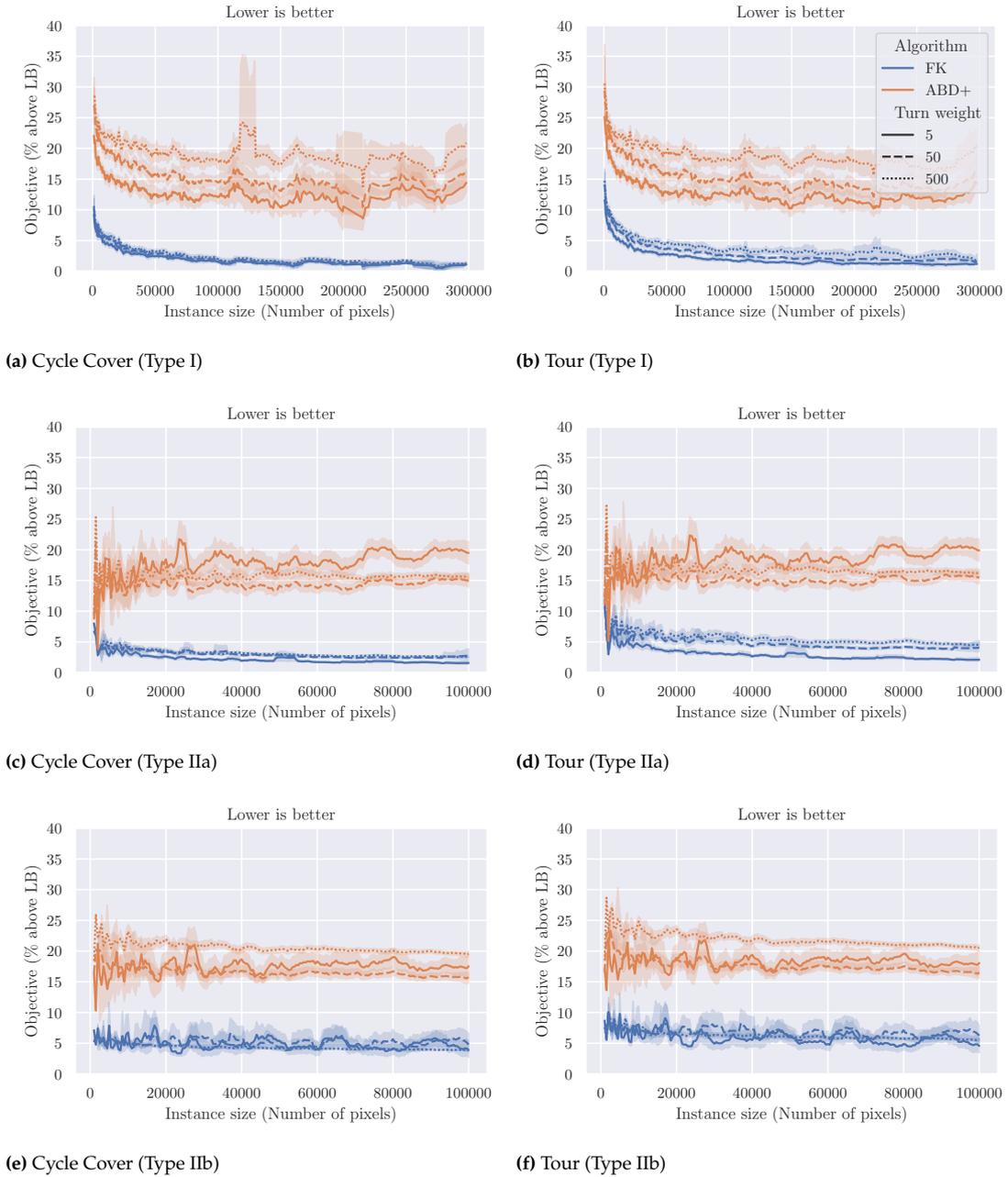


Figure 5.15.: Relative performance of the approximation methods for very large (but simple) instances of Type I and large (and more complex) instances of Type IIa and Type IIb. For examples of these instances see Figure 5.11, Figure 5.12, resp. Figure 5.13. The objective function is a linear combination of c times the number of simple turns, plus the number of pixel transitions, for $c = 5, 50, 500$. The lower bound is provided by the fractional solution. The difference between the cycle cover version and the tour version is in most cases insignificant, because the cycles are usually relatively large, so the cost of connecting the cycles is negligible compared to the costs of the cycles themselves. There are at least 1000 instances per instance type, for Type I even 3000.

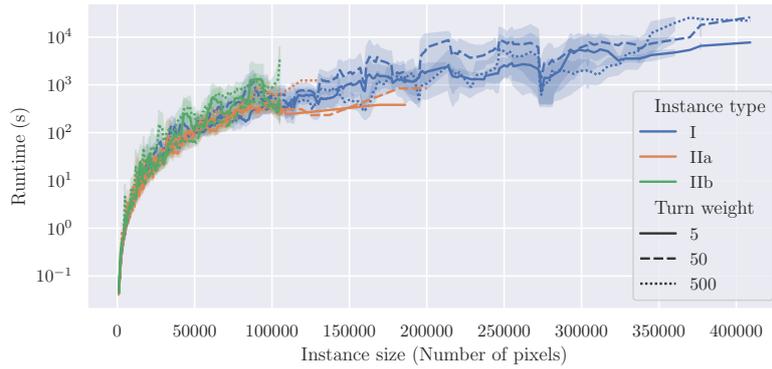


Figure 5.16.: Runtime of the cycle cover approximation algorithm for Type I, Type IIa, Type IIb and different turn cost coefficients, indicating the relative cost of a simple turn vs. a pixel transition.

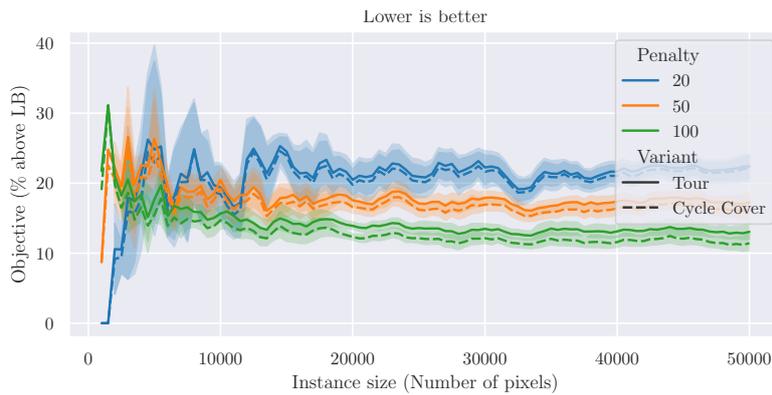


Figure 5.17.: Performance of the penalty variant of the new approximation algorithm on instances of Type IIa. The cost of a simple turn is 500 \times that of a pixel transition, while the penalty for missing a pixel is 100, 50 or 20 times the cost of a pixel transition. The ‘pockets’ at the boundaries make instances of this type particularly challenging, because paying the penalty instead of performing expensive detours is a non-trivial alternative.

pixels are worth skipping. The penalties are uniform for all pixels to make the analysis easier, but the implementation also supports individual penalties.

5.5. Conclusion

This chapter showed how we can compute near-optimal solutions even for large instances with 300 000 pixels of tour planning problems with turn costs in grid graphs. By carefully replacing the critical tasks of the original algorithm by equivalent but much smaller and faster tasks, we transformed a theoretically polynomial but impractical algorithm to a well scaling practical algorithm while maintaining the quality guarantees. The used techniques are not only much faster but also much simpler than, e.g., complex primal-dual optimizations of the original algorithm. As a consequence, practical problems of precision farming and pest control that were previously relying on local heuristics without any performance guarantee can now rely on well-understood algorithmic techniques with excellent performance guarantees. This shows the power of algorithm engineering techniques, demonstrating that new theoretical algorithmic insights can be turned into important practical breakthroughs.

There is a considerable range of practical future developments. These include adapting the techniques to further practical problems, in which covering tours are one component of integrated optimization problems involving several types of agricultural robots. In addition, new progress in precision farming (which aims at making use of refined data describing

intricacies of plant growth and ground yield) will make it relevant to adjust subset and penalty tours according to changing situations. We can be optimistic that demonstrating the practical benefits of algorithm engineering will be helpful for future interdisciplinary collaboration in the context of digital agriculture.

The next chapter builds on the proposed techniques of this chapter, but extends them to more complex instances beyond regular grid graphs. It also confirms the near-optimality of this approach for more complex and irregular grid graphs.

Generalization to Polygonal Areas

6.

This chapter implements a coverage tour optimizer for polygonal environments considering the robot's movement dynamics, and allowing partial coverage focused on important areas. We analyze how to convert the area into a well-fitting grid or mesh, and we adapt the approximation algorithm for regular square grids of the previous chapter to work on arbitrary meshes. After adding additional optimizations, this results in a powerful framework to optimize practical coverage trajectories with a theoretical foundation.

6.1. Introduction

Coverage path planning is an important problem for many different applications such as aerial surveillance [130], cleaning [131], milling [132], mowing [133], and more. It has already received a considerable amount of attention, mostly from a practical perspective, but also with some theoretical results. The problem is provably hard to solve on multiple levels, as it contains NP- and PSPACE-hard problems such as the TRAVELING SALESMAN PROBLEM (TSP), COVERING, and the PIANO MOVER PROBLEM.

The probably simplest theoretical abstraction of the problem is the TSP IN GRID GRAPHS. Here, we simply place a regular grid over the area and compute the shortest tour that visits all vertices at least once. The resolution of the grid is chosen such that visiting all vertices of the grid results in a (nearly) complete coverage. Because the TSP appears in many applications, it is probably one of the most well researched optimization problems, such that there are highly capable solvers despite its proved hardness. The Concorde solver [134] is able to solve instances with thousands of vertices to proved optimality [135] and there are other algorithms that can compute reasonably good solutions for much larger instances. Concorde is also used to optimize coverage paths, e.g., by Bormann et al. [131]. However, TSP in Grid Graphs is too strong of a simplification of the involved dynamics. The solutions, even the optimal ones, are often expensive to follow because the turns induce significant costs despite their short length. This is addressed in the problem MILLING WITH TURN COSTS, which not only minimizes the length but also the sum of turn angles the tour performs through the grid [33]. Unfortunately, turn costs increase the complexity of the problem such that not only itself but already the cycle cover relaxation becomes NP-hard [18]. While we were able to increase the size of optimally solvable instances from less than 100 vertices [115] to over 1000 vertices [5]¹, the still large difference to classical TSP shows the limits of computing optimal solutions for realistic dynamic models, even for strongly simplified environments.

Besides complex dynamics, we sometimes do not need to cover the whole area. A true 100% coverage is in many cases even not achievable because the tool simply does not fit into every corner. Instead, we have a feasible area that allows us to move in, and a smaller subset of it that is actually

6.1	Introduction	109
6.2	Problem Definition	111
6.3	Related Work	112
6.3.1	Full Coverage	113
6.3.2	Partial Coverage	113
6.3.3	Touring Costs	114
6.4	Generalized Algorithm	115
6.4.1	Discretization	115
6.4.2	Fractional Solution	117
6.4.3	Atomic Strips	118
6.4.4	Matching	121
6.4.5	Local Optimization	122
6.4.6	Connecting Cycles	124
6.4.7	Local Optimization	126
6.4.8	Default Algorithm	128
6.5	Grids and Meshes	128
6.5.1	Regular Grids	129
6.5.2	Meshes	134
6.5.3	Comparison	138
6.6	Evaluation	142
6.6.1	Integralization	143
6.6.2	Local Optimization	143
6.6.3	Optimality Gap	146
6.6.4	Objectives	146
6.6.5	Runtime	148
6.7	Conclusion	150

1: This was achieved by using a different formulation and concentrating on efficient integral constraints to eliminate subtours. Because turn costs actually minimize subtours, it can be advantageous to only add subtour-elimination constraints for (nearly) optimal solutions. This approach is not only faster but also simpler to implement.

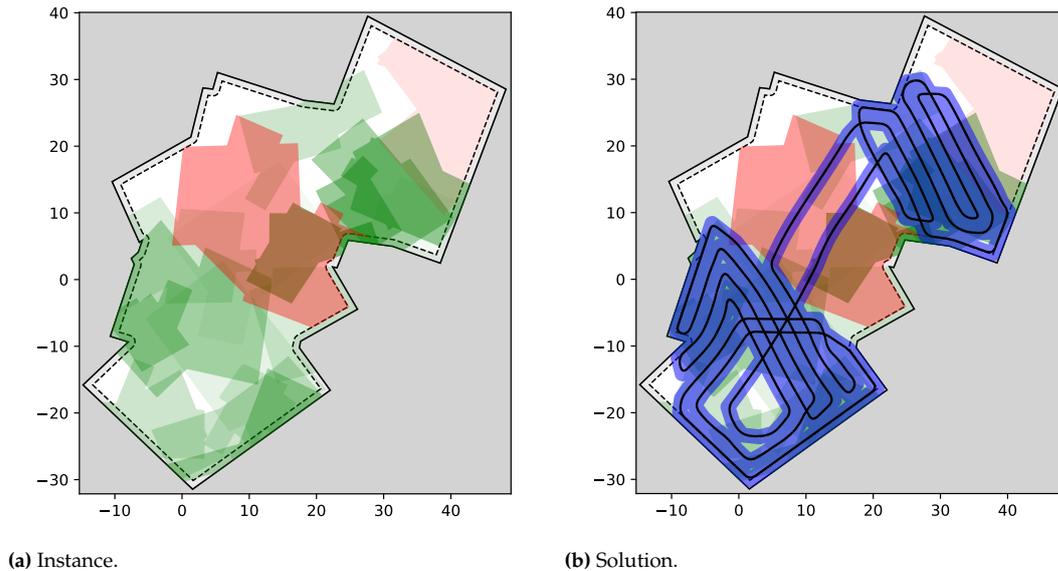


Figure 6.1.: Example instance and solution of partial coverage with a circular tool. The instance consists of a feasible area (polygon with dashed line denoting the feasible positions), valuable areas (green) that yield a reward when covered, and expensive areas (red) that multiply the touring costs consisting of distance and turn costs. We search a trajectory (black) that maximizes the covered value (covered area highlighted in blue) and minimizes the touring costs. In this example, the valuable area within the expensive area in the middle is barely covered because it is not worth the costs. The expensive area is only crossed twice to connect the two larger valuable areas.

‘valuable’. A vacuum robot can move within the whole room, but often there are dirt-prone areas and cleaner areas, which do not need to be cleaned every time. A harvester can move along the whole field, but crop yield can be heterogeneous; the harvester does not need to harvest everything, rather only most of the harvest. For aerial supervision, there are areas of higher and lower interest. Additionally, there may be areas that are harder to pass than others. For example there can be a fluffy rug that slows down the vacuum robot, a muddy area that hinders the harvester, or inhabited areas that should be avoided by an unmanned aerial vehicle (UAV). The algorithm of the previous chapter is able to optimize for partial coverage only in regular square grids, which do not represent real environments.

In this chapter, we generalize the algorithm of Chapter 5 to deal with arbitrary polygonal instances. We are given a polygonal area in which we can move with a circular tool or robot that covers everything within a fixed radius r . The feasible area contains all feasible positions, as we need to stay at least r away from the boundary. Additionally, we have valuable polygonal areas that reward us on (partial) coverage, and expensive polygonal areas that increase the touring costs within. An example with a solution can be seen in Figure 6.1. A solution is a tour that minimizes the missed value (the part of the valuable areas that is not within a distance of r to the tour) as well as the touring costs. If only half of a valuable area is covered, we miss half its value. The touring costs are a linear combination of distance and turn-angles, which are potentially multiplied if the part of the tour is within expensive areas.

In the following section, we first give a formal description of the problem in Section 6.2. This definition is purely for mathematical clarity and can be skipped by most readers. Afterwards, we give an overview of the related work and support for the used model in Section 6.3. In Section 6.4, we

describe how to generalize the approximation algorithm of the previous chapter. Understanding the approximation algorithm beforehand is useful, but not necessary as all steps are extensively visualized to give an intuition on what is going on. In Section 6.5, we study and evaluate different grids and meshes to transform a polygonal instance to a graph-based instance. This part may be of interest if you want to design your own grid-based coverage path planning algorithm. In this case, you can jump directly to that section. Finally, we consider the influence of our optimizations in Section 6.6, and the quality of our solution on the grid, independent of the grid quality, in Subsection 6.6.4. As usual, the chapter is closed with a conclusion and summary in Section 6.7.

6.2. Problem Definition

We evaluate our optimization approach on a simplified, but still generic two-dimensional geometric model, which we define in this section. This model can be adapted to many realistic scenarios, and many specifications are not due to algorithmic limitations but only used to simplify the evaluation. While a simulation based evaluation would yield more realistic results, it would be less generic and require a large set of realistic instances, which is hard to come by.

Let us first discuss how we model the robot. In the following, we primarily speak of robots, but generally all kinds of tools like milling machines or UAVs are included. We model the robot as a circle of radius $r > 0$, and its position $p \in \mathbb{R}^2$ is defined by its middle point. The robot immediately covers everything below it, i.e., if it is at position p , $\text{Cov}(p) = \{p' \in \mathbb{R}^2 \mid \|p - p'\| \leq r\}$ is covered. This makes the robot rotational invariant and simplifies many computations. The circular coverage may seem unrealistic at first glance for, e.g., a mower; but in a tour, the coverage of a line perpendicular to the trajectory is nearly identical to that of a circle.

The environment, e.g., walls or obstacles, can restrict the robot's movement. We denote the *feasible area* $F \subset \mathbb{R}^2$ as the set of all feasible positions of the robot and approximate it by a (non-simple) polygon. In the examples and evaluations, we start with a larger polygon representing, e.g., a room, and shrink it by removing the parts too close to the boundaries. F does not need to coincide with the coverable area, which allows us to separate the robot's shape from its coverage.

We define the trajectory T of the robot as a closed chain of waypoints $w_0, w_1, \dots, w_{|T|-1} \in F$. The robot moves in straight lines between the waypoints. We denote the corresponding segments by $\text{SEGMENTS}(T) = w_0w_1, w_1w_2, \dots, w_{|T|-1}w_0$ and demand that all segments $s \in \text{SEGMENTS}(T)$ are fully contained in the feasible area F . The trajectory in the following part is also called *tour*, in accordance with the previous chapter. An intermediate solution of multiple (closed) trajectories that still have to be connected to a tour is called a *cycle cover*, and its elements are *cycles* or *subtours*.

Additional to the feasible area F , we have *valuable areas* and *expensive areas*. Valuable areas $\mathbb{Q} = Q_0, Q_1, \dots \subset \mathbb{R}^2$ with weights $t(Q_i) \in \mathbb{R}^+$ represent the parts we want to cover. Expensive areas $\mathcal{E} = E_0, E_1, \dots \subset F$

with weights $m(E_i) \in \mathbb{R}^+$ represent areas with increased touring costs. Both types of areas are again approximated by polygons to simplify the computations.

The objective is to compute a feasible tour that maximizes the coverage value and minimizes the touring costs. To combine maximizing the coverage and minimizing the costs, we convert the maximization of the coverage into a minimization. This is achieved by considering the opportunity loss, i.e., the value of the missed area.

We use $|Q|$ to denote the size (area) of a polygon Q and the pure Q to denote the set of all contained points.

$$\begin{aligned} \min_T \text{COVERAGELOSS}_{\mathbb{Q},r}(T) + \text{TOURCOST}_{\mathcal{G}}(T) \quad (6.1) \\ \text{s.t. } s \subseteq F \quad \forall s \in \text{SEGMENTS}(T) \end{aligned}$$

The advantage of this objective over others is that its lower bound is zero, which allows a better comparison. We define the coverage loss and touring costs in the following.

Let $C_r(T) = \{p \in \mathbb{R}^2 | \exists s \in \text{SEGMENTS}(T), p' \in s : p \in \text{Cov}(p')\}$ denote the covered area of a tour. Note that the inclusion of a point in a segment is defined as lying anywhere on the segment and not just on its endpoints. This allows us to define the coverage loss formally by the maximally achievable coverage value minus the actually achieved value.

$$\text{COVERAGELOSS}_{\mathbb{Q},r}(T) = \sum_{Q \in \mathbb{Q}} |Q| \cdot t(Q) - \sum_{Q \in \mathbb{Q}} |Q \cap C_r(T)| \cdot t(Q)$$

The touring costs consist of weighted distances and turn angles.

$$\text{TOURCOST}_{\mathcal{G}}(T) = \lambda_0 \cdot \text{DISTCOST}_{\mathcal{G}}(T) + \lambda_1 \cdot \text{TURNCOST}_{\mathcal{G}}(T)$$

The two weights $\lambda_0, \lambda_1 \geq 0$ allow to weight the distance and tour costs, and we vary them in our experiments. Let $\mu_{\mathcal{G}} : \mathbb{R}^2 \rightarrow \mathbb{R}^+$ define the cost multiplier at a tool position, which allows us to model local cost changes induced by the environment. It is computed by $\mu_{\mathcal{G}}(p) = \prod_{E \in \mathcal{G}, p \in E} m(E)$.

The distance costs are now defined as

$$\text{DISTCOST}_{\mathcal{G}}(T) = \sum_{s \in \text{SEGMENTS}(T)} \int_{p \in s} \mu_{\mathcal{G}}(p) dp \quad (6.2)$$

For $\mathcal{G} = \emptyset$, this becomes $\sum_{s \in \text{SEGMENTS}(T)} \|s\|$. The turn costs only occur at waypoints and are also subject to the multiplier.

$$\text{TURNCOST}_{\mathcal{G}}(T) = \sum_{i=0}^{|T|-1} \mu_{\mathcal{G}}(w_i) \cdot \text{TURN}(w_{i-1}, w_i, w_{i+1}) \quad (6.3)$$

$\text{TURN}(p_0, p_1, p_2)$ denotes the turn angle at p_1 while traversing $p_0 \rightarrow p_1 \rightarrow p_2$. The indices of the waypoints are taken modulo $|T|$ to form a cycle.

6.3. Related Work

This section provides related and previous work on coverage path planning from a theoretical and practical perspective. We focus on the areas of

the classical full-coverage, as well as variants that allow a partial coverage, as in our model. Additionally, we provide an overview of previous work on more complex touring cost estimations, especially including turn costs. Our touring cost model, based on experimental evaluation, was strongly motivated by previous work on drones. Additional related theoretical work can be found in Section 5.1.1 (Related Work) on page 96.

6.3.1. Full Coverage

Planning a trajectory for a tool to cover an area, e.g., mowing a field or vacuuming a room, is known as the *Coverage Path Planning* problem (CPP). The CPP already enjoyed a lot of attention for different applications, models (e.g., multi-robot), constraints, and objectives, as can be seen in multiple surveys [130, 131, 136, 137]. There are multiple approaches, the two most prominent being: (1) decomposing the larger area into simpler areas that can be covered using spiraling or zig-zag patterns ([138–140]) and (2) applying a (regular square) grid onto the area, where each grid cell roughly represents the coverage area, converting the geometric coverage problem into a discrete touring problem on grid graphs ([141–145]). In this chapter, we (again) use the second approach but try to fit the grid to the polygonal area, sacrificing its regularity. Finding a good grid can be challenging but is fundamental for this approach, as we see later in this chapter. When only considering the length of the trajectory, the problem becomes the famous TRAVELING SALESMAN PROBLEM (TSP). Even in regular square grids, it is NP-hard [146] but there exist highly engineered solvers like Concorde [135] that are able to solve TSP-instances with 85 900 vertices to provable optimality. The TSP is theoretically and practically very well researched; we refer the curious reader to [147]. When adding turn costs, which is advisable for an accurate model of reality as we see soon, the problem gets more complicated. Even previously simple relaxations become NP-hard [18] but constant-factor approximations are available [33]. Recently, we were able to increase the size of instances that can be solved to optimality within minutes from around 100 vertices [115] to over 1000 and nearly optimal instances to even over 300 000 vertices [5], also described in Chapter 5 (Engineering an Approximation Algorithm) on page 95. This chapter utilizes this approach but applies it to more complex, irregular grids.

For general graphs, tours including turn costs are incredibly hard. If the turn costs correspond to geometric turn-angles, the problem is known as the ANGULAR METRIC TSP. The currently best known approximation algorithm by Aggarwal et al. [34] only achieves a factor of $O(\log n)$. For more general turn costs, it is known as the QUADRATIC TSP and also plays an important role, e.g., in bioinformatics [148].

6.3.2. Partial Coverage

We consider the problem of partially covering the area, based on weighted subareas, allowing the tour to focus on the important areas and skip less important areas.

Murtaza et al. [142] compute a full-coverage of the area, but prioritize subareas based on a probability distribution to find targets quickly.

Sharma et al. [144] also compute a full-coverage of the area, but with a limited budget, resulting in multiple tours that try to efficiently cover as much as possible. Both consider only distance costs in a square grid.

Jensen et al. [149] and Soltero et al. [150] perform coverage without a fixed radius, but minimize the distance of (weighted) points of interests to the trajectory. Jensen et al. [149] do this under an energy budget constraint that also considers the number of turns, while Soltero et al. [150] minimize a combination of the length of the tour and the distance of the tour to points of interests. This model is feasible for some inspection or surveillance problems and allows to create tours with varying resolution, but is not feasible for tools, like lawn mowers or cleaning robots, with strict radii.

Papchristos et al. [151] and Ellefsen, Lepikson, and Albiez [152] consider partial inspection of three-dimensional structures with distance and turn costs. Papchristos et al. [151] try to maximize the inspection reward of weighted sites subject to time constraints. However, the focus is less on covering area and more on touring a subset of sites. Ellefsen, Lepikson, and Albiez [152], on the other hand, focus on the surface of a single object with a multi-objective optimization for maximizing the covered surface while minimizing the needed energy. While both problems have a strong resemblance to our problem, the coverage models are very different and are not directly applicable to our targeted scenarios.

6.3.3. Touring Costs

A reasonable amount of work on covering considers models with distance and turn costs in various degrees, such as only minimizing the number of turns [149], the sum of turn angles [141, 145, 151] (like us), or even model- and experiment-based cost functions [138, 153].

Cabreira et al. [153] and Modares et al. [145] independently performed experiments to evaluate the real energy costs of multicopters. Modares et al. [145] results indicate nearly linear costs for the length and for the turn-angle, resulting in a simple linear combination of travelled distance of a tour and the sum of turn angles, supporting our model. The energy consumption is also primarily based on how long the multicopter has to stay in the air. Cabreira et al. [153] also show a nearly linear necessary deceleration for turns but only starting at 50°, below which no deceleration is performed. Incorporating this into our model, however, is dangerous, as every larger turn could be divided into small turns and would need additional constraints. Contrary to the findings of Modares et al. [145], the straight distances in [153] did not have linear costs as the drone can accelerate more on larger distances, see also the prior work of Di Franco and Buttazzo [154]. Linear distance costs are still a reasonable approximation, considering that the maximum velocity is often quickly reached.

The costs are often not homogeneous but can change within an area due to, e.g., wind fields for UAVs [155] and difficult terrain or inclinations [133] for ground-based vehicles. However, only few consider this for CPP, e.g., [133, 143]. More can be found for simple path planning [156–158]. Location-dependent cost modifiers also allow modelling of soft obstacles that can be crossed if necessary, but induce a penalty and should be

avoided. We allow such modelling by denoting polygonal subareas in which the costs are multiplied by specified factors.

Our approach computes an ordered set of waypoints which includes turns, that are too hard to be efficiently executed directly by many robots. These, however, can be smoothed: Nam et al. [159] (using cubic interpolation), Artemenko et al. [160] (using rational quadratic Bézier curves), or Forsmo et al. [161] (using a more involved MIP-formulation that considers a lot of the dynamic). The optimal approach depends heavily on the dynamics of the robot, and is thus skipped here.²

2: The visualizations in this chapter use a Bézier curve-based smoothing to make the tours look more natural, but the coverage computations are not smoothed.

6.4. Generalized Algorithm

In the previous chapter, Chapter 5, we engineered an approximation algorithm to compute nearly optimal solutions for instances with 300 000 pixels in regular square grids. In this section, we show how to adapt this algorithm to solve polygonal instances including expensive and valuable areas. More precisely, we show how to approximate the area using an embedded graph, and we adapt the previous algorithm to work on arbitrary embedded graphs.

The generalized algorithm has seven steps:

1. Discretization: Convert the polygonal instances to a discrete graph that approximates the area. We call this graph a grid or mesh, based on its properties.
2. Fractional Solution: Compute a fractional solution in this graph using linear programming.
3. Atomic Strips: Select atomic strips, including the dominant one, using the fractional solution. This step is more complicated for general meshes than for regular square grids, which can be solved as discussed in Chapter 5.
4. Matching: Perform a matching on the atomic strips and obtain a cycle cover.
5. Local Optimization: Improve the cycle cover by heuristics.
6. Connecting Cycles: Connect the cycles to form a tour.
7. Local Optimization: Improve the tour by heuristics.

We now consider these steps in detail. Additional evaluations of the performance are performed in the sections thereafter.

6.4.1. Step 1: Discretization

Before we can apply our approximation technique, we have to convert the polygonal area into a graph of potential waypoints. Instead of a complex geometric problem, we then just have to find a tour in a graph where each vertex yields some coverage, and the touring costs are based on the used edges and edge transitions.

The simplest and most common strategy is to place a regular square grid over the feasible area. The points and edges that are fully contained, become our graph. This would also directly allow us to use the algorithm of the previous chapter. However, this is not always optimal. Other options are to use regular triangular grids or irregular generated meshes.

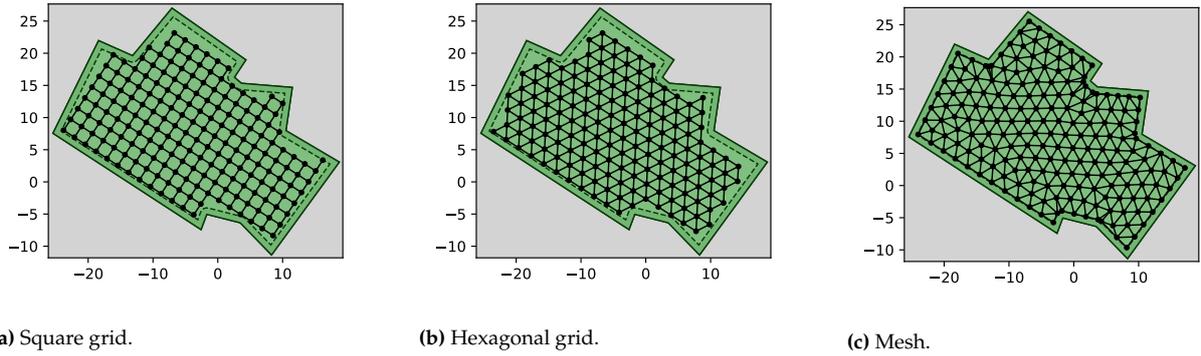


Figure 6.2.: Different grids for transforming a polygonal instance to a graph instance that can be solved with a variation of the approximation algorithm. The green area is the area to be covered but due to the robot's radius, we can only place waypoints inside the dashed area. We can rotate a regular grid to fit especially the interior area nicely. Alternatively, we can use an irregular grid created by a meshing algorithm. It can better adapt to the shape, especially the boundary, of the area but its irregularity can also make it more expensive for coverage inside the polygon.

To keep the computational costs low, it is generally better to have fewer vertices and a low edge degree at the vertices. Not only the computational costs increase, also the quality of the relaxation decreases at vertices with more neighbors. Examples with various grids can be seen in Figure 6.2.

Computing the edge costs and the turn costs at the vertices is straightforward and can directly use the definition in Section 6.2. The graph is just a subset of the actual solution space, and so we can simply precompute the costs of the individual parts used by the graph. The value of covering a vertex is more complicated. We can simply assign the value that the robot covers when being at this point, but this can easily over- or underestimate the real value. It can underestimate the value if the real coverage actually happens when moving to and from the vertex. It can overestimate the value if other vertices are close by, and the coverage areas are overlapping. Generally, it is good if the sum of values in the graph also equals the maximal value in the original instance. We could simply scale all the values to achieve this but as the graph can have a heterogeneous distribution, using a Voronoi diagram is a better option. The Voronoi diagram is a classical method from Computational Geometry which partitions the area such that each vertex gets the area assigned closest to it. Using the value of these areas gives us a value assignment that equals the original values and is sensitive to the neighborhood of the vertices. This can be seen in Figure 6.3.

In the following, we denote the resulting graph by $G = (P, E)$ and call P (potential) waypoints. Every tour T on G consists of segments in E , which are fully contained in the feasible area, and is, thus, a feasible tour in the original polygon. We denote the coverage value assigned to a waypoint $p \in P$ by $\text{val}(p)$, and it corresponds to the coverage value of p 's Voronoi-cell. The distance cost of an edge $pp' \in E$ is defined by $\text{dist}(p, p') = \int_{x \in pp'} \mu_{\mathcal{E}}(x) dx$, according to Equation 6.2. The turn cost of passing p through the neighbors n and $n' \in N(p)$ is defined by $\text{turn}(n, p, n') = \mu_{\mathcal{E}}(p) \cdot \text{TURN}(n, p, n')$ according to Equation 6.3.

Obtaining a good graph is a fundamental problem, and the whole Section 6.5 (Grids and Meshes) on page 128 is focussed on it.

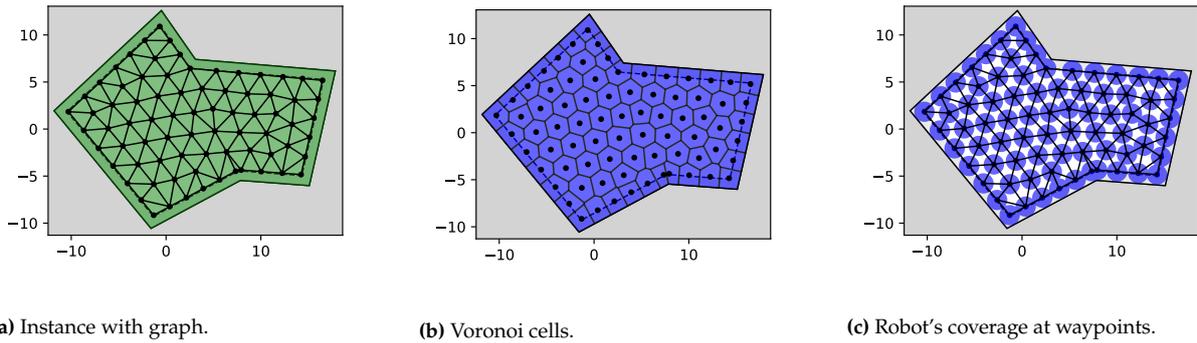


Figure 6.3: Computing the value of a vertex, especially when using irregular grids, should use Voronoi-diagrams or similar techniques to obtain a nice partition of the area, as seen in (b). Using only the area covered at the corresponding waypoint yields under- and overestimation due to ignored area that would indirectly be covered by using edges and intersecting ranges, as seen in (c). The corresponding instance with the graph is shown in (a). Underestimating the value of a waypoint can result in the algorithm skipping it. Overestimating the value of a waypoint can result in the algorithm including it at a high cost.

6.4.2. Step 2: Fractional Solution

Now that we have a discrete graph with weights and values, we can obtain a fractional solution by using linear programming. Examples for fractional solutions covering the whole area or for partial coverage are given in Figure 6.4 resp. Figure 6.5. This is nearly identical to the original approximation algorithm and provides us with good orientation hints for the next step in Subsection 6.4.3.

Given the graph $G = (P, E)$, we work on passages $uvw = wvu$ that cover a waypoint $v \in P$ coming from or going to the neighbored waypoints $u, w \in N(v)$. For every passage uvw , the variable $x_{uvw} \geq 0$ denotes how often the passage is used. The cost of using the passage is defined by $\text{cost}(uvw) = \text{turn}(u, v, w) + 1/2 (\text{dist}(u, v) + \text{dist}(v, w))$. We use only half the distance, because otherwise the distance would be doubly charged by the two incident passages. Additionally, we use the variable $s_v \geq 0$ that denotes skipping the waypoint and consequently eliminating its value.

$$\min \sum_{v \in P} \text{val}(v) \cdot s_v + \sum_{u, w \in N(v)} \text{cost}(u, v, w) \cdot x_{uvw} \quad (6.4)$$

$$\text{s.t.} \quad \sum_{u, w \in N(v)} x_{uvw} + s_v \geq 1 \quad \forall v \in P \quad (6.5)$$

$$2 \cdot x_{wvw} + \sum_{u \in N(v), u \neq w} x_{wvu} = \quad (6.6)$$

$$2 \cdot x_{vww} + \sum_{u \in N(w), u \neq v} x_{vuw} \quad \forall vw \in E$$

The objective in Equation 6.4 simply minimizes the missed coverage value and touring costs. Equation 6.5 enforces a waypoint either to be covered or skipped, and Equation 6.6 enforces a consistent flow, i.e., every edge is used equally from both sides.

We can perform a few optimizations if the value of a waypoint $v \in P$ is very high or very low. Whenever the value of a waypoint is zero, the constraint in Equation 6.5 that enforces coverage as well as the skipping

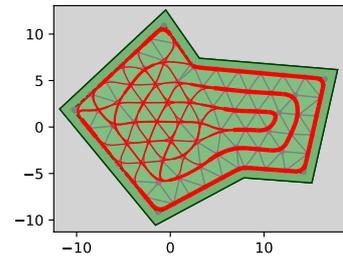


Figure 6.4: Fractional solution in red for full-coverage. The line thickness denotes the fractional value. For some areas, the fractional solution already provides an integral tour.

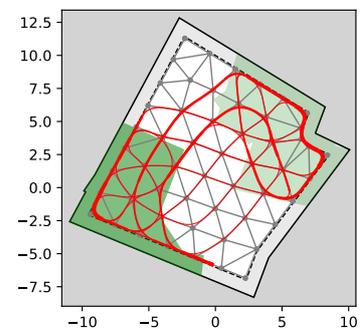


Figure 6.5: Fractional solution in red for partial coverage. The line thickness denotes the fractional value. The valuable areas are in green (the thickness represents the value).

variable are unnecessary. If the value is very high, i.e., higher than the touring cost of a small cycle over a neighbor, s_v can be directly fixed to zero because it is too valuable to be skipped.

Of course we could already try to connect fractional subtours to form a tour. This is more complicated than for the classical TSP for multiple reasons:

- ▶ Waypoints do not have to be visited at all.
- ▶ Waypoints can be visited multiple times.
- ▶ There can be cycles that visit the same waypoint, but are not connected.

We describe MIP-constraints in [19, 27]. Due to a Big-M notation in them, the corresponding relaxation is rather weak and not very useful for fractional solutions. Additionally, the next steps of our algorithm try to optimize a cycle cover first, and prematurely optimizing for tours could actually harm the next steps.

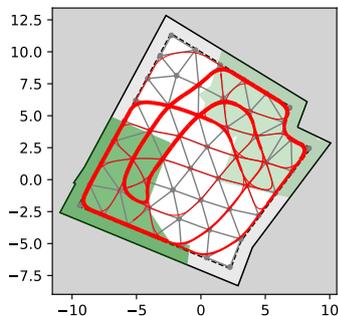


Figure 6.6.: The fractional solution of Figure 6.5 after some branch and bound iterations to improve the relaxation.

A feasible optimization is to improve the LP-relaxation by performing some branch and bound steps, thereby making it more integral. This is also performed by MIP-solvers like Gurobi, but with a different focus. While Gurobi primarily wants to compute an integral solution (which would take too long), we just want to achieve a more integral relaxation. The improved relaxation is just a by-product of Gurobi. We can tell Gurobi to focus on improving it, but unfortunately there does not seem to be a way to access it. Luckily, this part is reasonably easy to implement ourselves, and the only difficult decision is to decide on the variable to branch on. We always work on the least expensive leaf and return it after a specified number of branching steps. Gurobi would also perform some cutting planes, like Gomory-cuts [162], which we discard because of their complexity to integrate. Note that continuing to optimize a model after adapting the variable bounds is much faster than the initial run because a lot of the work can be reused (also called warm-start). An example can be seen in Figure 6.6, where we applied this approach to the example in Figure 6.5. The variable to branch on is selected by

$$\operatorname{argmax}_{uvw} \operatorname{cost}(uvw) \cdot \operatorname{frac}(x_{uvw}) \cdot \sum_{n,n' \in N(v)} \operatorname{frac}(x_{nn'})$$

where $\operatorname{frac}(x) = |x - \operatorname{round}(x)|$ denotes the difference to the closest integer. The idea is to prioritize expensive waypoints that have many non-integral passages, or forks, and force them to decide for one. We will look into this optimization again in Subsection 6.6.1.

6.4.3. Step 3: Atomic Strips

In this step, we convert the instance such that we can use a minimum-weight perfect matching to compute an integral cycle cover, i.e., a solution, that is allowed to consist of multiple tours. Without turn costs, an optimal cycle cover can actually be computed in polynomial time because the costs of the edges are independent. With turn costs, the cost of an edge depends on orientation of the preceding edge, making the problem NP-hard even in grid graphs [18, 27]. We use the fractional solution of

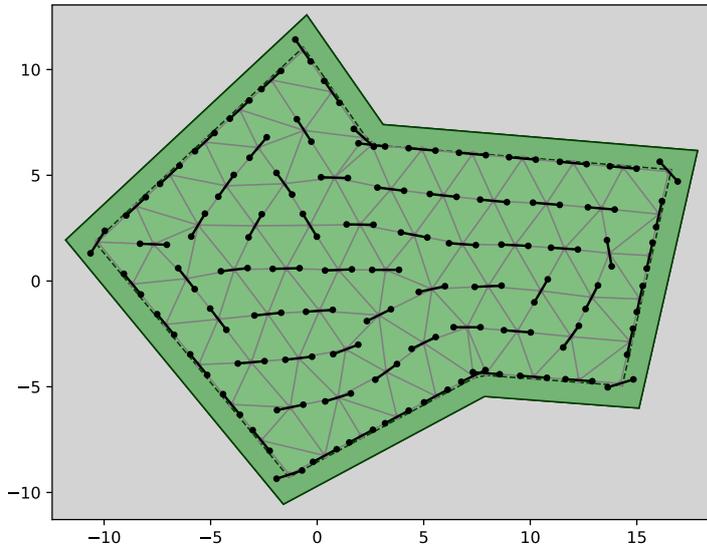


Figure 6.7.: Replacing every waypoint by an atomic strip (black segments) converts the problem into a matching problem without losing the turn costs. The orientation of each atomic strip needs to be guessed from the fractional solution (Figure 6.4), and wrong guesses can degrade the solution.

the previous step to predict the corresponding orientation and make the costs independent again.

We can imagine this procedure as replacing every waypoint by an epsilon-length segment, as in Figure 6.7. Computing a minimum-weight perfect matching on the endpoints, as in Figure 6.8, yields the optimal cycle cover that includes all these segments. The necessary turns at the joints are fixed for every connecting edge, and can therefore be accounted for in the edge weights together with the distance. We are calling these epsilon-length segments *atomic strips*. The possibility of skipping a waypoint can be implemented by adding an edge between the two endpoints of its atomic strip with the weight of the missed coverage.

The orientations of the atomic strips are of fundamental importance: If we guess them correctly, the minimum-weight perfect matching actually corresponds to an optimal cycle cover on the waypoints. If we guess the orientation of an atomic strip badly, the minimum-weight perfect matching may perform expensive turns to integrate it.

Luckily, the exact orientation is less important if we make turns at a waypoint, as the range of optimal orientations increases with the turn angle, as shown in Figure 6.9. For a u-turn, every orientation is optimal. The straighter a passage, the more important a good orientation becomes; but often these cases are easy to guess from the fractional solution, which is why this approach works so well in Chapter 5.

This observation allows us to limit the orientations to the orientations of incident edges, i.e., neighbors. In the following, we represent the available orientations for the atomic strip of a waypoint $p \in P$ by the adjacent waypoints $N(p)$. If the atomic strip of p has the orientation $n \in N(p)$, one of its endpoints heads at n .

A waypoint may need to be crossed multiple times, as we are limited to passages within the grid $G = (P, E)$. This can easily be implemented by transitive edges, i.e., two successive edges uv and $vw \in E$ automatically create an edge uw with the combined costs. However, we learned in Chapter 5 that introducing *optional* atomic strips and only allowing

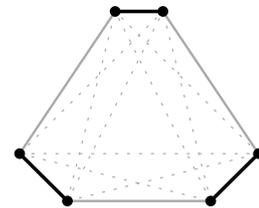


Figure 6.8.: Segments (black) can be connected to a cycle cover via a minimum-weight perfect matching (gray) on the end points. Because of the fixed joints, we can charge the turn costs to the edge weights.

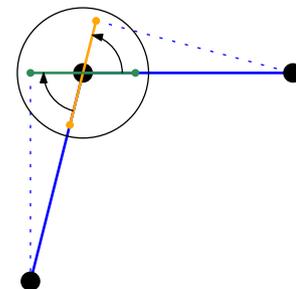


Figure 6.9.: The orange and the green atomic strips represent the turn equally well. Only the assignment of the turn costs to the weight of the matching edges changes. Also, all atomic strips in between are equally good.

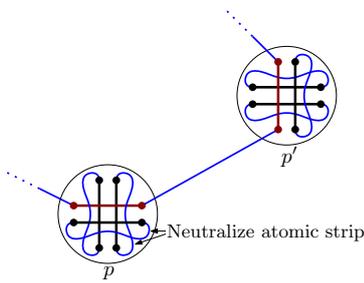


Figure 6.10.: The waypoints p and p' each have two horizontal and two vertical atomic strips. The tour induced by a matching (blue) only uses one atomic strip of each, and skips the other by edges connecting both endpoints. These edges are usually zero-weight, except for one carefully-selected one that has the weight of the opportunity loss $\text{val}(p)$ resp. $\text{val}(p')$.

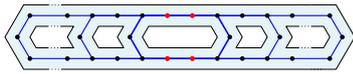


Figure 6.11.: Optimal tours with turn costs in a regular triangular grid can require a linear amount of passages through some waypoints (red).

direct connections scales much better. An optional atomic strips can be implemented by simply adding an edge with zero weight between its endpoints, see Figure 6.10. We call the non-optional atomic strip of a waypoint the *dominant* one.

In a square grid as in Chapter 5, we obtain a 4-approximation if we add an atomic strip for every neighbor and declare the most used one in the fractional solution as dominant. In a triangular grid, there can be waypoints that are passed a linear number of times, as in Figure 6.11, but this is an artificial instance. In our instances, every waypoint is usually only covered once or twice. As every atomic strip increases the computational complexity, we limit the number of atomic strips to a constant k , and allow for every waypoint $p \in P$ at most one atomic strip per neighbor $n \in N(p)$. The task is to select a subset $A \subseteq N(p)$ with $|A| \leq k$ as atomic strips and determine the dominant one.

If $k \geq |N(p)|$, we can simply choose $A = N(p)$. This allows us to use any passage twice without overhead, because any waypoint passage has either two neighbors with each having an optimal atomic strip or the passage is a u-turn. If $k < |N(v)|$, things get more complicated because we want to optimize three often opposing objectives:

- ▶ We want to improve the expected case, i.e., the passages with the highest likelihood should be as cheap as possible.
- ▶ We want to minimize the cost overhead of the average case, i.e., the average overhead of any passage.
- ▶ We want to minimize the worst case, i.e., the cost of the worst unexpected passage.

Our strategy for this case consists of two phases. First, we select atomic strips based on edge usages in the fractional solution. This optimizes the expected case. Second, we fill up the remaining atomic strips by minimizing the sum of squared overheads of passages not used in the fractional solution. This optimizes the average and worst case scenarios (using a higher exponent would shift the focus to the worst case).

The precise strategy is given in Algorithm 2; $\text{FS}(v, w) = \sum_{u \in N(v)} x_{uvw}$ denotes the usage of the edge $vw \in E$ in the fractional solution, and $\text{OH}(uvw, A)$ denotes the minimal overhead if the passage uvw has to use an atomic strip in $A \subseteq N(v)$. The overhead corresponds to the additional turn costs needed to accommodate a (possibly misaligned) atomic strip.

If a waypoint $v \in P$ has a coverage value, i.e., $\text{val}(v) > 0$, we still have to select the dominant strip that can only be skipped at the cost of the opportunity loss. For very straight passages, there may be no atomic strip that can be used without overhead (this can also be due to numeric issues). Thus, we propose a more dynamic approach.

The selection of the dominant strip a out of the set A is performed by the *usage* of the atomic strips, as defined in the following function:

$$\text{SELECTDOMINANT}(v, A) = \operatorname{argmax}_{a \in A} \sum_{u, w \in N(v)} x_{uvw} \cdot \text{USAGE}(uvw, a)$$

Let $\text{TURN}_a(u, v, w)$ be the turn angle if the passage uvw is forced to use the atomic strip a . The usage depends on the turning overhead induced

Algorithm 2: Atomic strip selection**Input:** A waypoint v and number k .**Output:** A selection of at most k atomic strips, identified by $N(v)$.

```

1  $A \leftarrow \emptyset$ 
2  $C_0 \leftarrow \{n \in N(v) \mid \text{FS}(v, n) \geq \varepsilon\}$ 
3  $C_1 \leftarrow N(v) \setminus C_0$ 
  /* 1. Select by usage in fractional solution.          */
4 while  $C_0 \setminus A \neq \emptyset$  do
5    $A \leftarrow A + \text{argmax}_{v \in C_0 \setminus A} \text{FS}(v, w)$ 
6   if  $|A| = k$  then return  $A$ 
  /* 2. Select by overhead.                              */
7 while  $C_1 \setminus A \neq \emptyset$  do
8    $A \leftarrow A + \text{argmin}_{n \in C_1 \setminus A} \sum_{u, w \in N(v)} \text{OH}(uvw, A) \cdot \text{OH}(uvw, A+n)^2$ 
9   if  $|A| = k$  then return  $A$ 
10 return  $A$ 

```

by forcing a passage to use it. If there is no overhead, the usage is 1.0, and if there is overhead, the usage drops exponentially.

$$\text{USAGE}(uvw, a) = \lambda^{(\text{TURN}_a(u, v, w) - \text{TURN}(u, v, w)) / \phi}$$

λ is the usage at an additional turning angle of ϕ , see Table 6.1 for an example. Higher values allow more gap, which is necessary if the grid is not regular. We use $\lambda = 0.25$ and $\phi = 45^\circ$ in the experiments.

An example for different k can be seen in Figure 6.12.

Future Work 6.1.

There are many possible parameters for selecting the atomic strips, and the quality of the selection is essential for the quality of the following matching. Can we devise a better strategy, e.g., by the usage of reinforcement learning?

Overhead	Usage
0°	1.00
1°	0.93
2°	0.85
5°	0.68
10°	0.46
30°	0.10
50°	0.02

Table 6.1.: Usage for $\lambda = 0.1$ and $\phi = 30^\circ$

6.4.4. Step 4: Matching

We are left with a weighted graph on the endpoints of the atomic strips, and we want to compute a minimal matching. There are edges between any endpoints of atomic strips belonging to neighbored waypoints in the grid. The weight corresponds to the touring costs between the two waypoints, with the corresponding orientation at the endpoints. Additionally, each atomic strip has an edge between its two endpoints. For the dominant atomic strip, the weight corresponds to the opportunity loss, i.e., the assigned coverage value, when not covering it. For all others, the cost is zero to allow skipping them without additional costs. Let k be the maximal number of atomic strips at a waypoint, then the number of vertices and edges in the matching instance is in $O(|P| \cdot k^2)$.

We solve the corresponding minimum-weight perfect matching instance with the Blossom V algorithm of Kolmogorov [128]. The author states a worst-case complexity of $O(n^3m)$, which would be prohibitive, but in practice it shows to be extremely fast.

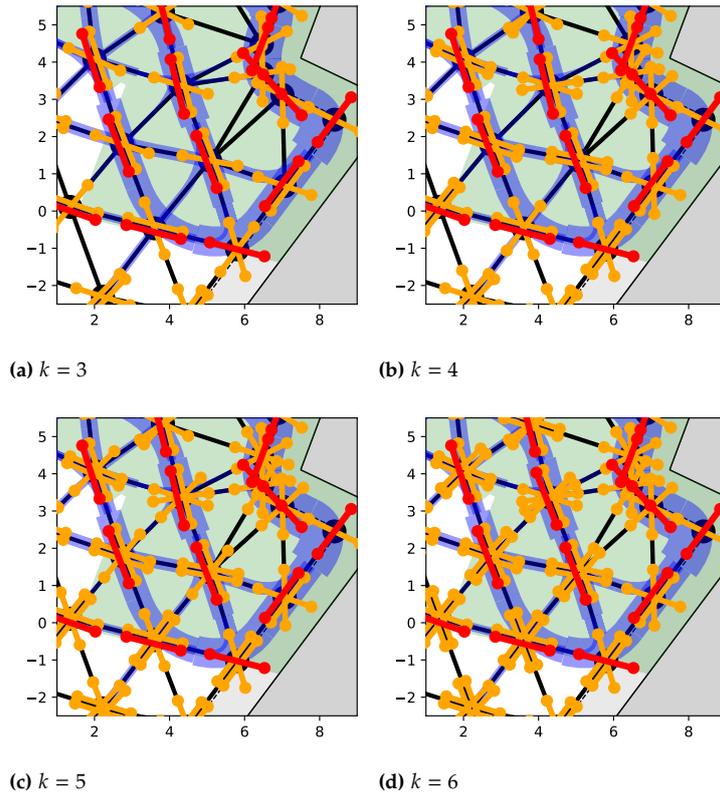


Figure 6.12.: Example of atomic strip selection for different k . The atomic strips are displayed in yellow (optional) and red (dominant). The grid is displayed in black and the fractional solution in blue. First, we select the atomic strips by their usage in the fractional solution, then by minimizing the overhead. The dominant strip is selected purely based on usage.

Connecting the atomic strips via the matched endpoints yields a set of cycles, see Figure 6.13, that we can connect in the next step. Contrary to the previous steps, there are not any direct options for quality improvements. Only the runtime can be improved by further engineering the matching algorithm or further simplifying the graph. Because we are focussing on solution quality, we wont do further engineering of this step, even though it is computationally the most complex and limiting step.

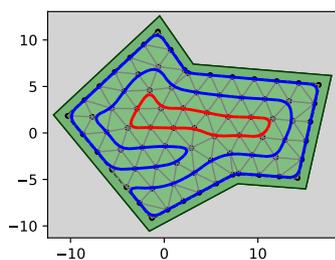


Figure 6.13.: The matching of the atomic strip of Figure 6.7 yields a set of tours. In this case, a red and a blue cycle. It can also directly decide not to cover some waypoints, but in this case the coverage values are very high.

Future Work 6.2.

Currently, we fully connect the atomic strips of two adjacent waypoints. This results in $\Theta(k^2)$ matching edges for any edge in the underlying grid, with k being the number of atomic strips per waypoint. Many of these matching edges represent expensive turns, and are unlikely to be used or even highly redundant. For example, a u-turn can use any atomic-strip with the same overhead. Can we reduce the number of matching edges to improve the runtime while maintaining the solution quality?

6.4.5. Step 5: Local Optimization

Before we continue to connect the cycles to a single tour, we can optimize the cycle cover. For this, we select a small but expensive part of the solution and compute a (nearly) optimal solution via mixed integer programming. This can be repeated multiple times until a satisfying solution is obtained, as shown in Figure 6.14. Note that we are able to solve instances with 1000 vertices in regular square grids to optimality,

as described in [5, 19, 27]. Also, for irregular grids, small instances with less than 100 vertices can usually be solved within seconds. We denote the desired number of vertices for local optimization by t .

We select the expensive area to be optimized by choosing an expensive root and selecting the first t vertices of a breadth-first-search. The expense of a waypoint in a solution is denoted as the cost of the passages covering it, or the corresponding opportunity loss if it is not used. To make the selection more robust, we also include the expenses of all direct neighbors by summing them.

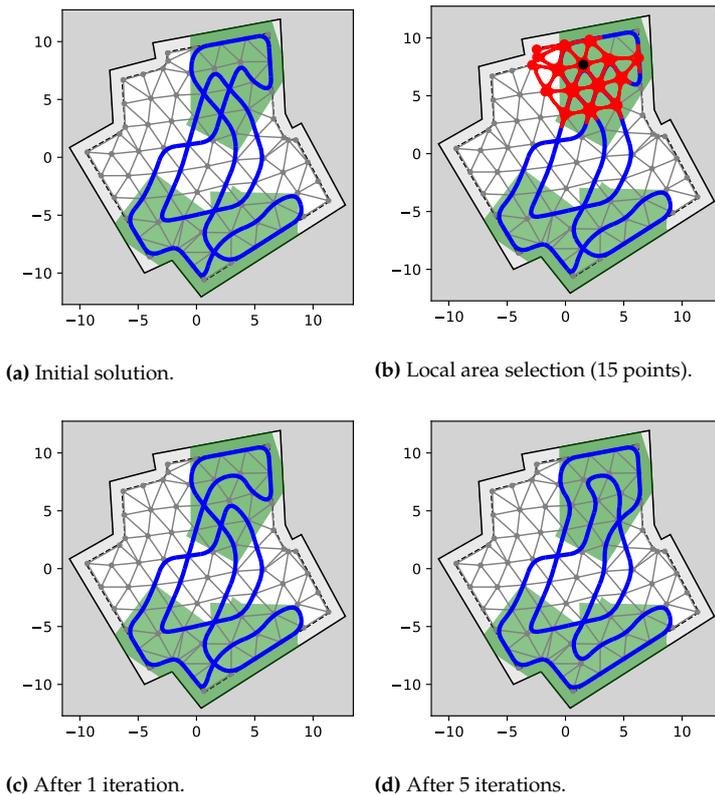


Figure 6.14.: By optimizing local areas (red) of cycle covers (blue) with mixed integer programming, we can improve the initial cycle cover. The solution provided by the previous approach without optimizations is shown in Figure 6.14a. We then select an expensive area (Figure 6.14b) and optimize it to near optimality, resulting in (Figure 6.14c). After five such iterations, we end with a visibly improved solution (Figure 6.14d). By chance, the optimized solution is even connected.

By simply replacing the fractional variables with integral variables, the linear program in Section 6.4.2 (Fractional Solution) on page 117 yields a corresponding MIP. In this MIP we fix all variables of the given solution except the variables corresponding to the $t + 1$ selected waypoints. Of course, we do not need to include the fixed waypoints in this MIP at all but only need to place the corresponding constants into Equation 6.6. This ensures that the local solution remains consistent with the fixed exterior solution. After optimizing the local MIP, we replace the part in the solution and exclude the root and its neighbors to be selected as root in further iterations. This is necessary because the expensive parts can already be optimal (within their local area).

A useful property of the MIP is that the optimization process usually is faster, if our (local) solution is already (nearly) optimal. If we provide the MIP-solver with the corresponding start solution, it only has to find a matching lower bound. Using the running time and the actual improvements, one could improve the selection of the next area, or dynamically increase it. By choosing disjunct areas, this optimization approach also allows efficient parallelization. However, we leave such

optimizations to future work, and simply perform i iterations for a fixed area size t . See Subsection 6.6.2 for an analysis of this approach.

Future Work 6.3.

Due to the matching technique, we can only use turn costs that depend linearly on the turn angle. The MIP used in this optimization technique can use any turn cost function. If we approximated a non-linear turn cost function with a linear one, we could now use the original cost function (we can also use it for the fractional solution).

- ▶ Can we compute good coverage paths for tools with non-linear turn costs?
- ▶ Can we adapt this optimization technique to efficiently prevent very sharp turns that are infeasible for some machines?

This also applies for the tour optimization in Subsection 6.4.7.

6.4.6. Step 6: Connecting Cycles

Now, we only need to connect the cycles to form a tour. For adjacent cycles, this is quite simple and involves only minimal extra costs: simply go through every edge that connects two cycles and perform a merge via the least expensive one, see Figure 6.15. A simple optimization would be to use two parallel edges once, instead of one edge twice, but this is also done automatically in Subsection 6.4.7.

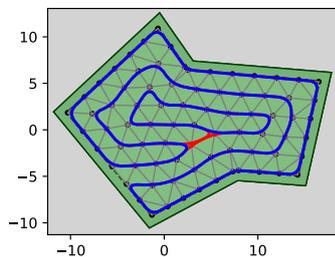


Figure 6.15.: Connecting the two cycles of Figure 6.13 via an edge (red).

Things get more complicated if the cycles are farther apart. It is valid to question if this would be worthwhile. It could be that the connection costs actually outweigh the touring costs of the corresponding cycles. If the area covered by the cycle is not valuable enough, we are better off simply removing the cycle. We see such a scenario in Figure 6.17: We have two remote valuable areas with separate cycles. However, the valuable area on the right side does not actually have much value, and we therefore only connect the two cycles on the left. If we increase the value of the area on the right, it suddenly becomes advantageous to connect all cycles.

To select any cycles, we first need to know how much each cycle is worth. We estimate the value of a cycle by the sum of values of its covered waypoints. If a waypoint occurs in different cycles, only the first cycle gets its value. This can happen if two cycles cross and cannot be connected due to turn costs. Because this rarely happens, the estimated cycle values are reasonably accurate if the coverage values of the waypoints is accurate.

Next, we need to know how expensive it is to connect any two cycles. This can be achieved with a modified Dijkstra-algorithm on the edge graph. Working on the edge graph of the grid allows us to include not only the distance of the path, but also the turn costs between any two edges. To make thing simpler, we use a directed version where we also include the direction through which we pass the edge, as can be seen in Figure 6.16.

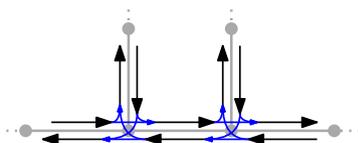


Figure 6.16.: Converting the grid (gray) into a directed edge graph to compute a shortest path with turn costs inside. The distance and turn costs are assigned to the blue arcs.

The distance cost of using an edge can now simply be assigned to the outgoing arc in the edge graph. If we let k be the maximum degree in the grid, then we have at most $O(|P| \cdot k)$ vertices and $O(|P| \cdot k^2)$

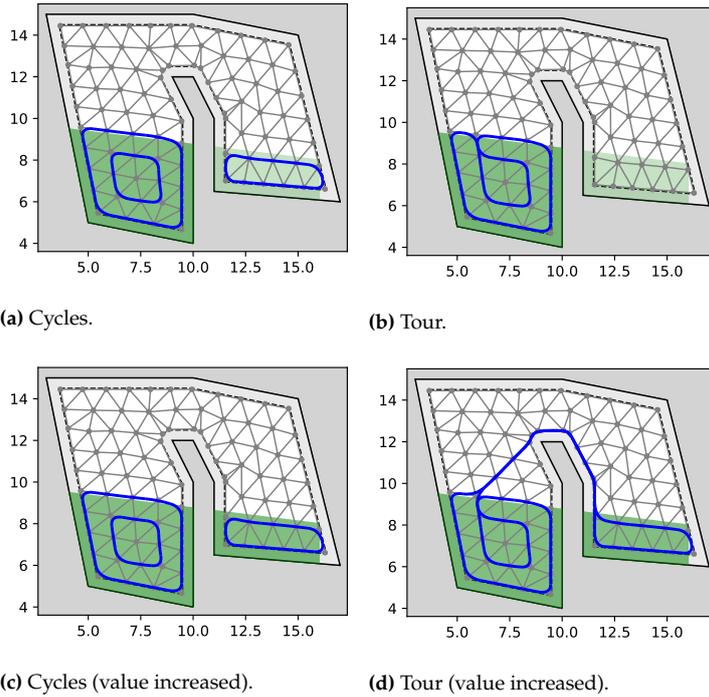


Figure 6.17.: If the valuable areas (green) are more distanced, the cycles (blue) should only be connected if the value is high enough in relation to the costs. A prize-collecting steiner tree is used for this.

edges in the auxiliary graph. Using Dijkstra's algorithm, we can compute the least expensive path between any two edges (ignoring possibly collected coverage value) in $O(|P| \cdot k^2 \log |P|)$. The costs are symmetric, so it is optimal in both directions. Still missing are the costs involved merging a (doubled) path with a cycle. It would be expensive to check all combinations for edges incident to the two cycles. Instead, we can select one of the cycles and initialize all incident edges in the Dijkstra algorithm with the final connection costs to it. We now only have to find the least expensive incident edge to the target circle using the already computed distances by Dijkstra's algorithm. Hence, for every pair of cycles, at most one execution of Dijkstra's algorithm is necessary.

With these two pieces of information, we can compute a prize-collecting steiner tree (PCST) on the cycles and their connections. The resulting tree corresponds to the worthwhile cycles and how to connect them. Computing an optimal PCST is NP-hard, but there exists a 2-approximation by Goemans and Williamson [163]. We use an implementation by Hegde, Indyk, and Schmidt [164] with a runtime complexity of $O(d|E| \log |V|)$, where d refers to the encoding size of the numbers. A benchmark is given in [165]. If the corresponding graph is small enough, we compute an optimal PCST using integer programming. If there are some zero or negative connection costs, we can directly connect the corresponding cycles before we compute the PCST. Using a PCST instead of just greedily connecting cycles potentially also integrates cycles that are not valuable enough on their own, but they become valuable in combination with other cycles.

Using the PCST, we now iteratively merge cycles (using the doubled paths computed using the Dijkstra-approach) in a depth-first search starting from an arbitrary cycle in the PCST. Whenever we merge two cycles, the path creates additional docking points that may be cheaper than the originally computed connecting paths. However, we do not need

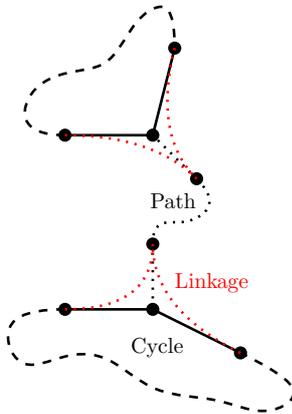


Figure 6.18.: Linking a (doubled) path with a cycle exchanges the incident passage in the cycle (unbroken line) by two that are half in the cycle and half in the path (dotted red).

to recompute the whole Dijkstra-tree, but can simply reduce the costs for the corresponding edges and let the reduced costs propagate. During the joining of the cycle with the doubled path, passages are actually replaced from the cycle, see Figure 6.18. Shortest paths originating from such removed passages become invalid. As this occurs rarely and can be detected, recomputation should only be performed if such an invalid shortest path is used.

Because the connection costs change during merging, it can result in better tours if we repeat the complete procedure after each merge and do not merge the complete PCST at once. While the runtime would still be polynomial, it would nonetheless be relatively expensive. Therefore, we won't make use of this optimization. The double usages of the involved paths can be easily optimized in the next step.

6.4.7. Step 7: Local Optimization

After connecting the cycles to form a tour, the connecting parts are often highly redundant, as can be seen in Figure 6.19. Luckily, we can extend the local optimization approach of Section 6.4.5 (Local Optimization) on page 122 to connected tours. The challenge this is to make sure that the tour remains connected after local optimizations. The used MIP does not enforce connectivity and may disconnect the tour again. A naïve approach is to only accept local improvements that preserve the connectivity and discard all others. This is of course quite restrictive and we can find a superior solution.

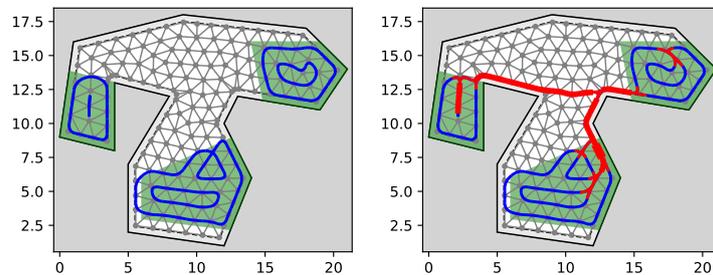


Figure 6.19.: Especially due to the connection approach of subtours, a lot of redundant coverages can be created, which we aim to minimize.

(a) Cycles before connecting.

(b) Tour with redundant parts (red).

Subtour elimination in the MIP is more difficult than for, e.g., the TRAVELING SALESMAN PROBLEM: not only are all visitations optional, but two tours can cross without being connected. Simply enforcing that two edges have to leave a connected component, therefore, does not yield the desired result. In [19, 27] we actually have a corresponding MIP. Because we already start with a tour and know that we have to connect an interior solution (inside the small area to be optimized) to the fixed exterior solution, we can devise a simpler separation constraint.

There are two types of subtours: those that are completely within the area and those that are only partially within the area. We can only get an infeasible solution with subtours of the second type if the local solution incorrectly connects the exterior solution. However, both types can be handled equally.

We either want a subtour C to dissolve or to become part of the connected tour. For this, there needs to be a vertex passage of a subtour to be unused, or a vertex passage leaving the subtour used. We select an arbitrary vertex passage of the first type and demand that the sum of the second type is greater than it. Note that this assumes the existence of an external, fixed solution, and is otherwise not exact.

Let X_A be the vertex passage variables that are contained in the area A and can be modified by the local optimization. This includes all variable x_{uvw} with $u, v, w \in A$. If u or w are not in A , vu resp. vw must be used in the solution, i.e., the edge connects the changeable interior solution to the fixed exterior solution. All other variables are fixed.

Let X_C be the vertex passage variables that are used by the subtour C . Let X'_C be the vertex passage variables that share one edge with the subtour C but are not in X_C . These are the vertex passages that leave the trajectory of C . We can now state a constraint that eliminates C , if it has been created by an optimization on X_C .

$$\sum_{x \in X'_C \cap X_A} x \geq x_c \quad x_c \in X_C \cap X_A, C \text{ is subtour} \quad (6.7)$$

There exist more efficient options for connecting, e.g., more distant subtours, but this hardly applies for optimizing only small areas. For the case that the MIP does not yield a connected solution for A within a fixed number of iterations, we leave A as it is. Applying this approach multiple times can significantly improve the solution, as can be seen in Figure 6.20. The individual steps are shown in Figure 6.21.

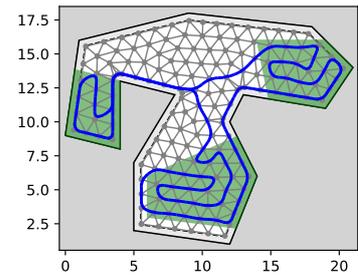


Figure 6.20.: Solution of Figure 6.19 optimized with 20 iterations of size 20.

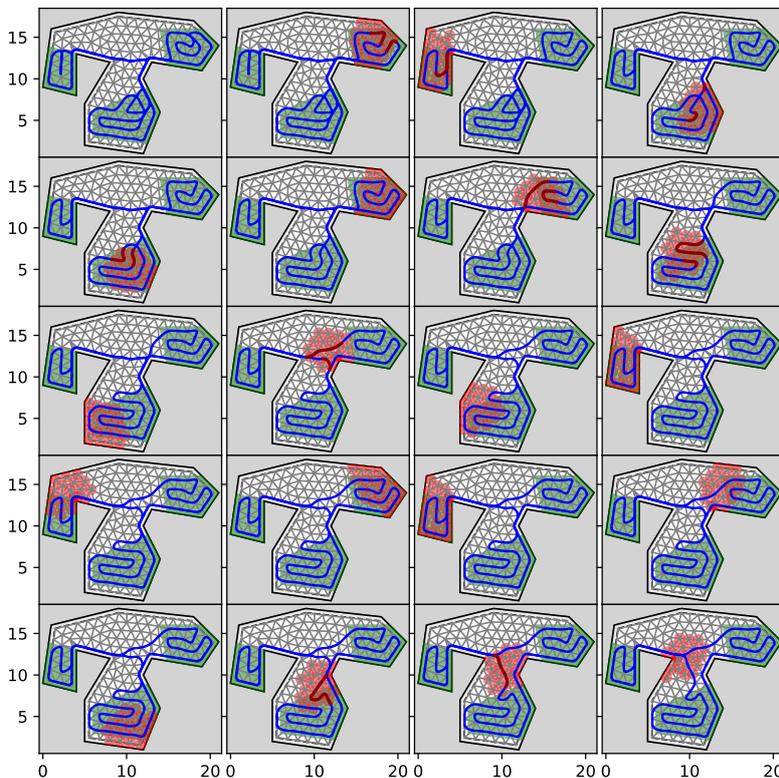


Figure 6.21.: Multiple steps of the tour optimization. The optimized area and the changed parts are highlighted in red. In some steps, no changes are made because the solution is (locally) optimal in the area.

Future Work 6.4.

The area selection is currently relatively simple and does not take into account the success or failure of previous steps. Can we determine effective areas for our optimization heuristic more adaptively?

Future Work 6.5.

The redundant vertex passages usually form a tree in which we should be able to efficiently compute the most expensive redundant path via dynamic programming. This expensive redundant path could then be replaced by a shortest path, which possibly also improves the coverage. The advantage of such an approach is the potential ability to optimize longer paths for which our local optimization heuristic would need many iterations. Can we use such an approach especially to optimize the redundant coverages created by the cycle connection algorithm?

6.4.8. Default Algorithm

For the following experiments, we use the following settings, if not stated otherwise:

- ▶ We perform 50 integralization steps to improve the integrality gap of the fractional solution.
- ▶ We select up to $k = 6$ atomic strips per waypoint.
- ▶ We perform 25 iterations of the cycle cover optimization with an area of 50 vertices.
- ▶ We first try to connect cycles greedily if the connection cost is zero or less. Otherwise, we compute an optimal PCST instead of using the approximation algorithm, because the corresponding graph is reasonably small for all instances.
- ▶ We perform 25 iterations of the tour optimization with an area of 50 vertices. If the resulting solution is not connected, we perform up to 10 cutting plane iterations.

The selection of the grid is more complicated, and it is dealt with in the next chapter. Because we only consider the number of iterations, the runtime can vary greatly. This is intentionally because the runtime of the LPs, MIPs, etc. vary anyway³, and a fixed number of iterations improves the comparability. Otherwise, a solver can be penalized just because a MIP took longer than for other solvers. For practical applications, dynamic parameters based on runtime and gradient are often better suited.

3: The runtime for a fixed instance is relatively deterministic with Gurobi, but the dynamic subproblems in the individual solver can differ strongly if the solvers use, e.g., different grids.

6.5. Grids and Meshes

An essential point for the grid-based methods is to choose a good grid. Choosing an unsuitable grid, or even just the wrong orientation for it, can drastically reduce the achievable performance as also noted by Bormann et al. [131]. Especially in the presence of turn costs, orientating a square grid in 45° to a straight boundary will result in many turns, because all vertices at the boundary will need to make a turn. Even if we

would be able to compute an optimal tour for this grid, the gap to the optimal solution independent of the grid can be nearly arbitrarily large, see Figure 6.22.

Choosing a good grid, hence, is as important as finding a good solution within this grid.

In this section, we focus purely on how to convert the polygonal instance into a graph-based instance. The performance of solving the graph-based instances with our approach is considered in the next section. The reason for separating these two parts is that this section can be applied to all grid-based coverage path planning algorithms. While we use our algorithmic approach for evaluation, most observations can be transferred to general coverage path optimizations. More specifically, we evaluate the following questions:

- ▶ How well do regular square and triangular grids perform in terms of touring costs and coverage?
- ▶ What is the optimal edge length in terms of touring costs and coverage?
- ▶ How can we create good triangular and quadrilateral meshes to approximate the area?
- ▶ How do regular grids and meshes perform for full coverage and partial coverage?

We start with regular grids, as many algorithms only support regular grids, and then continue to irregular grids, i.e., meshes.

6.5.1. Regular Grids

Before we go to the lawless meshes, let us take a look at regular grids.

The most common grid is the square grid, which essentially partitions the area into small squares. There are two options for the edge lengths, i.e., the distance between two adjacent vertices, for a circular tool with radius r : $2r$ or $\sqrt{2}r$. The first option reflects the optimal distances between two parallel trajectories, while the second option already provides a full coverage by simply visiting all vertices. In case of turns, a length of $2r$ will leave out a portion of the area, as can be seen in Figure 6.23.

Luckily, when minimizing turns, these cases are also minimized but not completely eliminated. The remaining cases can be fixed by slightly moving the waypoints, as shown to the right, but we leave this technique to future work. You can find an application of this technique for coverage without turn costs or constraints in the master's thesis of Perk [166].

Another common grid is the triangular grid. These are also known as hexagonal grids, because the dual graph consists of hexagons but the graph itself consists of triangles. Here, every vertex has six neighbors, which makes its optimization more challenging but also allows more complex turns. For achieving a full coverage by only visiting the vertices, we need a distance of $\frac{3}{\sqrt{3}}r$ to reach the center of each triangle. If we want two parallel trajectories to be perfectly apart, we need a distance of $\frac{4}{\sqrt{3}}r$. In this case, we again can lose coverage at turns, which we may need to fix.

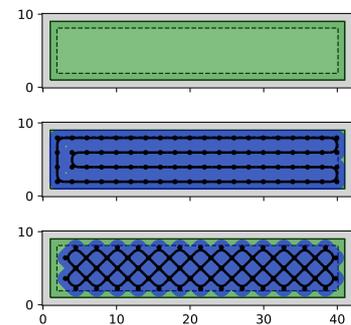


Figure 6.22.: An example that allows arbitrary high turn costs for an unsuitable alignments. The upper image shows the area to be covered in green. The middle image shows a well-aligned grid, and the lower image an unsuitable grid that requires many turns. The blue area is the covered area, and the black lines show a tour computed by the algorithm.

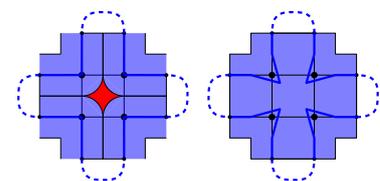


Figure 6.23.: With a distance of $2r$, the middle of a square is not fully covered (red). However, we minimize turns such that these scenarios are also minimized. If these scenarios happen, they can be fixed with reasonably small costs by moving the turning points slightly to the inside. Depending on how the subtours are connected, changing the connections can also be useful to obtain smaller turns.

Future Work 6.6.

What are the costs of closing the gaps in the coverage left by turns by slightly modifying the path like in Figure 6.23? Can we find a good strategy for triangular grids and even general meshes?

For square and triangular grids, we denote the denser version that covers the interior area at the vertices as *point-based*. The other version, which can miss area at turns, but covers everything between two parallel lines, as *line-based*. This results in four regular grids: *point-based square grids*, *line-based square grids*, *point-based triangular grids*, and *line-based triangular grids*. Examples with covered area from the vertices and tours can be seen in Figure 6.25. In those, we can also see that the greatest loss of coverage happens at the boundary. The point-based grids have a small advantage here because they can place more points due to their finer resolution. An easy solution to increase the coverage is to simply sweep once along the boundary, but this can be expensive for curvy boundaries. We will look into other solutions with irregular grids later.

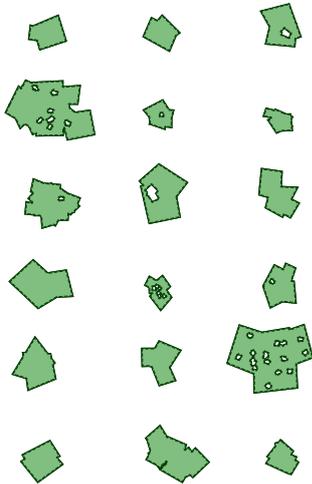


Figure 6.24.: Examples of the used (full-coverage) instances.

For comparing the different grids, we focus on polygons that can be reasonably well covered, i.e., that do not have narrow corners. We do not consider (integral) orthogonal instances, inspired by simple rooms, as these can clearly be converted to a square grid and pose no serious challenge. Instead, we create instances that are in the shape of more complicated architecture with many non-parallel lines. Additionally, we add obstacles for some instances. The instances are generated by merging multiple distorted rectangles and adding some holes with the same procedure. During this process, only steps that do not lead to narrow corners, bottlenecks, or even disconnection are chosen. By using a set of different random parameters for repetitions, sizes, and distortion strength, we generated a set of 200 instances. Examples of these instances can be seen in Figure 6.24. The weight for the turn costs is 1.0, 5.0, 50.0 (measured in radian), and the tool-radius is uniformly set to 1.0. For simplicity, we focus on full coverage and only compare the touring costs and the coverage (aiming for 100%). Partial coverage instances have many more parameters and are more difficult to compare. We can spare ourselves this complexity until Section 6.5.3 (Comparison) on page 138.

Because an unsuitable alignment can make most grids very expensive, we try 20 random alignments of each grid, plus one which we rotate such that the sum of the minimal passage costs per vertex are minimal. Of these, we choose the alignment with the minimal touring costs.

Let us first take a look into the touring costs on each of the grids. Of course they can differ for different algorithms, but it still gives a good indication of the quality of the grid. For comparing the touring costs of the different instances and solutions, we need to normalize the objectives. This is done by putting the objective in relation to the best of the corresponding instances. A value of 50% means that the tour is 50% more expensive than the least expensive solution. The plots in Figure 6.27 show that the *line-based* grids result in the least expensive tours by a significant margin of over 30%. The triangular grid has a small advantage of less than ~7% over the square grid. The point-based square grid performs worst and is on average 50% more expensive than the line-based triangular grid. The results are relatively stable also for larger instances. When comparing

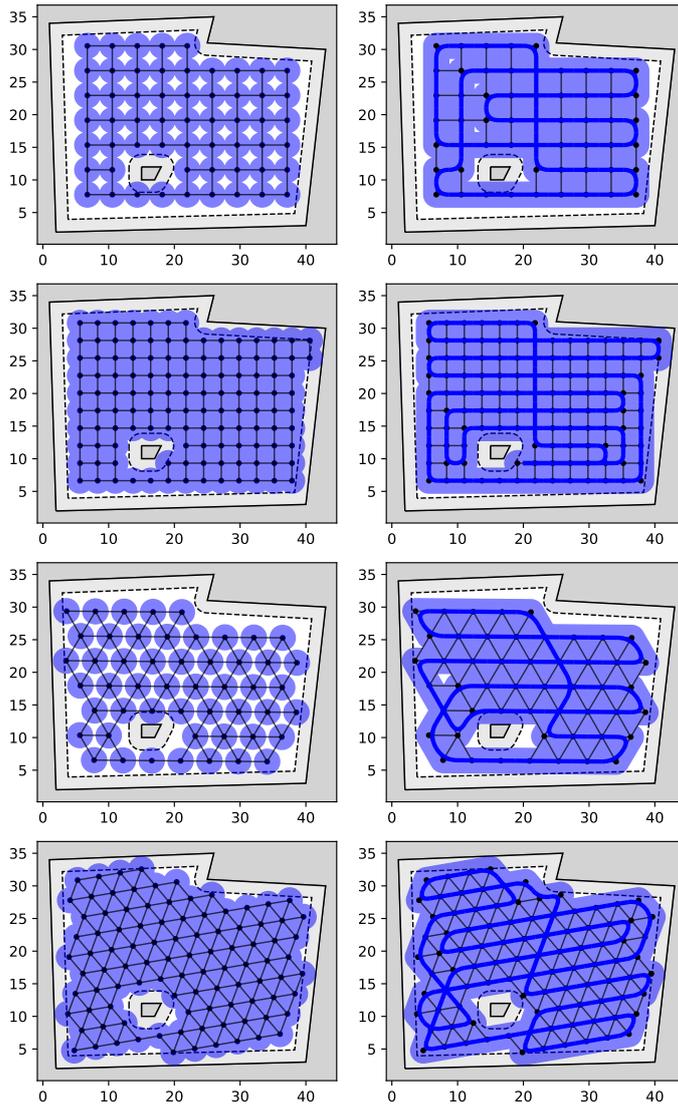


Figure 6.25.: Square and triangular grids with point-based (denser) and line-based (sparser) distances. The blue areas on the left indicate the covered area from the vertices of the grid. The blue areas on the right indicate the covered area of a corresponding tour (blue trajectory). While the line-based grids do have an insufficient coverage from the vertices alone, the tour only leaves few gaps at turns but is much shorter. The white area enclosed by the dashed boundary describes the feasible tool positions.

based on the turn cost weights, the triangular grids are very stable, while the square grids become slightly worse for higher turn costs. This can be explained by the reduced turning abilities (only having 90° and 180°) of the square grid.

Tours on line-based grids are, thus, less expensive than point-based grids, but how much does the coverage suffer? If the coverage is too reduced, the lower touring costs are of little comfort. Fortunately, the data in Figure 6.26 shows that the point-based triangular grid only covers on average less than 4.2% more than the line-based triangular grid. Considering also that the point-based grids miss a few percent of the coverage even for the larger instances, this is sufficient for many applications. The missed coverage that is especially high for smaller instances can be primarily attributed to the boundary. For larger instances, the boundary ratio gets smaller and, thus, the coverage gets better for all grids. In the end, we can decide between a 4.2% higher coverage using point-based grids, or a $\sim 25\%$ less expensive tour.

We selected the best alignment out of 20 random alignments and one optimized alignment, which is, of course, a deceptive selection. As

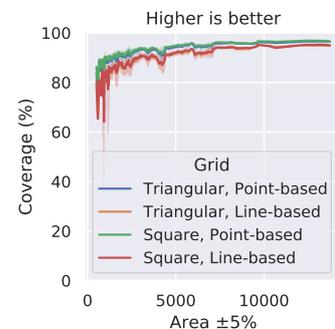


Figure 6.26.: The covered area of the various grid types. Point-based instances only have a small advantage.

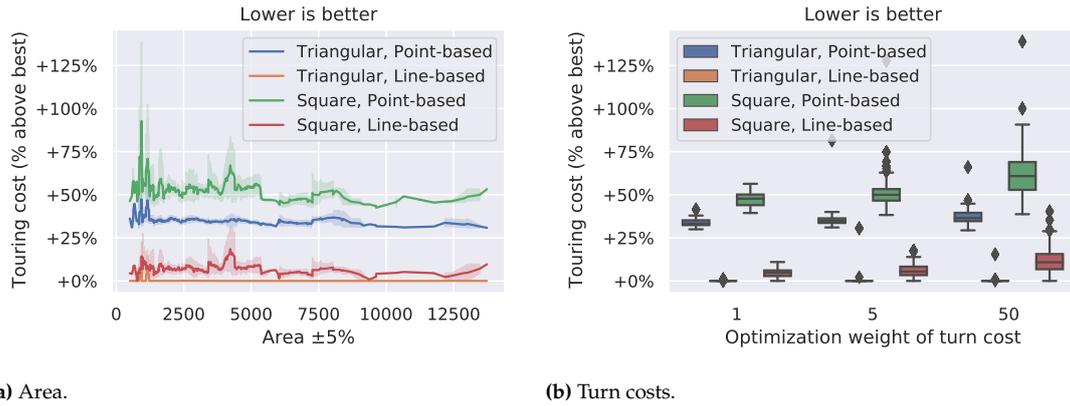


Figure 6.27.: Touring costs for the different regular grids. To counter unsuitable alignments, only the best alignment is used. The value shows how many percent the touring costs are on average higher than the least expensive tour among the solutions for the corresponding instance. Line-based grids yield clearly less expensive tours, and this effect is stable also for larger instances. The triangular grid only has a small advantage over the square grid of less than 7%. The right plot (b) shows that square grids become worse for higher turn costs.

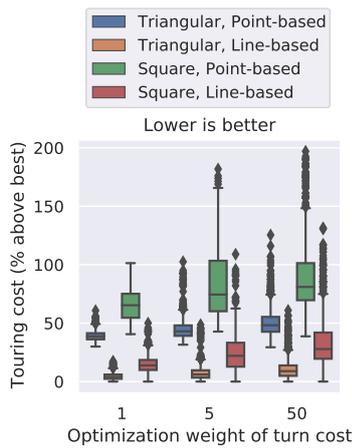


Figure 6.28.: Average touring costs for the random alignment compared to the best solution.

optimizing a tour in a grid takes some time, simply trying many random alignments is not very efficient. The natural question that comes to mind is, how well the average random alignment performs. This question can be answered by the plot in Figure 6.28: for some instances not very well.

The higher the weight of the turn costs, the worse is the average alignment. While for many instances, the random alignments of the line-based triangular grids are still reasonably good, there is a high deviation with many outliers. For the square grids, the random alignment is most of the time unsuitably aligned, implying that it needs a more careful alignment than triangular grids.

Examples for good, bad, and median alignments can be seen in Figure 6.29. The selection only considers the touring costs. The concrete instance has a high turn cost weight of 50, which leads to the high redundancy for the worse alignments, as the tour tries to only make turns at corner points where it has to make turns anyway.

For the experiments in the following sections, regular grids will be line-based triangular grids oriented by our heuristic. This heuristic, as also explained above, rotates the grid such that the sum of minimal passing costs for all vertices is minimal. The idea behind this is that we minimize the number of ‘stair-case’-boundaries that can be observed in the bad alignments in Figure 6.29. This heuristic performs reasonably well, and is on average only 2.7% more expensive than the best grid in the previous experiment. In comparison, the best of 20 random grids was only 0.6% more expensive, still showing some room for improvement for our heuristic. For the other grid types, the best of 20 random grids was also slightly better than our heuristic but only minimally (e.g., 10% vs. 9.2% for line-based square grids.). Overall, the heuristic works reasonably well and is much faster than computing the solution on 20 random alignments and returning the best solution.

Future Work 6.7.

One shortcoming of regular grids is that they often miss some area at the boundaries.

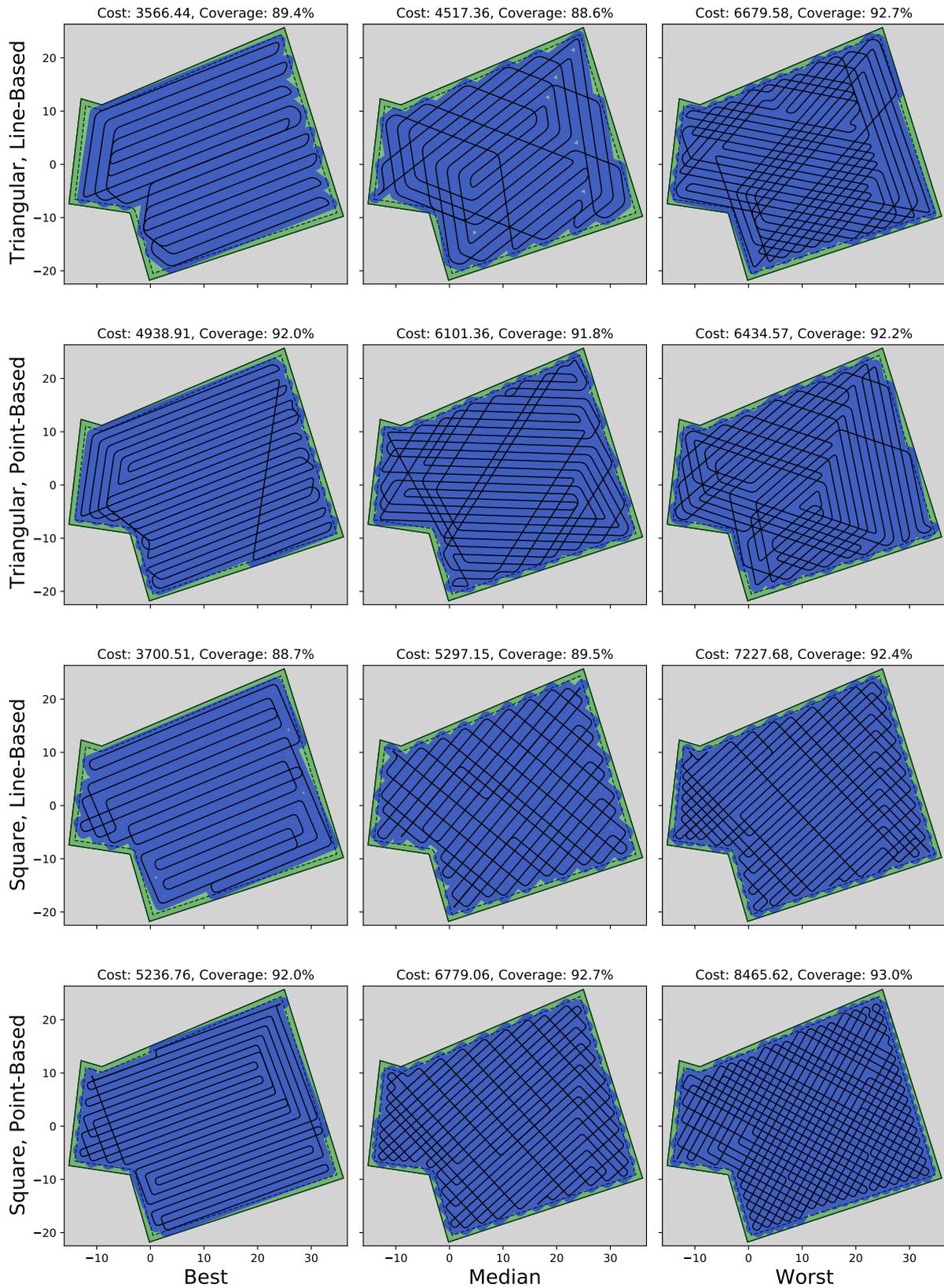


Figure 6.29: Examples for tours on the four grid types with different alignments. In the first column, the best alignment is shown. In the middle, an alignment that yielded an average tour; and on the right, the worst alignment for the grid type on this instance. The high weight on the turn costs produces lots of redundant straight coverages if the boundary is badly aligned to the grid.

- ▶ Can we mix the regular grid with a boundary mesh to improve the coverage at the margins while keeping the additional touring costs low?
- ▶ Can we use our algorithm to improve the coverage by putting a grid on the missed areas and merging the solution with a given tour?

This could allow us to use a highly optimized algorithm for regular (square) grids, as in the previous chapter, and only use our more generic algorithm for the difficult parts.

6.5.2. Meshes

We have seen how to approximate a polygonal area using a square or triangular grid. These regular grids are much easier to work with and are therefore very common for coverage path planning. However, the approximation can be very crude, not only missing large parts of the boundary but also being badly aligned to it. Luckily, we are not the only ones with the desire to approximate an area by a grid-like structure, and we can make use of the many results in the field of mesh generation. In this section, we consider the use of meshes instead of regular grid graphs.

The field of *mesh generation* (also known as grid generation, meshing, or gridding) considers the partition of a surface or geometric object into simpler elements, such as triangles or quadrilaterals (or three-dimensional counterparts). This is a fundamental task in computer graphics, physics simulation, geography, and cartography to deal with the more complex objects. The concrete specifications of a good mesh differ for some tasks, but important properties often are:

- ▶ Angles should not be too small or too wide. A primary motivation for this is due to numerical issues.
- ▶ The mesh should have a low complexity, i.e., minimize the number of introduced vertices and edges.
- ▶ The mesh should be as regular as possible, i.e., low variance in size and shape. Most algorithms allow the user to specify a desired or maximal edge length.

To find out more about the general subject of mesh generation, you can take a look into the surveys [167, 168] that are part of Computational Geometry books.

Many mesh generators are very sensitive to close points on the boundary and react by placing equally close points that result in a high density in this area, see Figure 6.30.

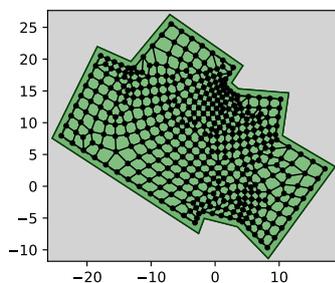


Figure 6.30.: A quadrilateral mesh that is sensitive to the close points on the boundary and reacts with a very dense grid in these areas. Generated with *gmsh* [169] using the *Frontal-Delaunay for Quads* algorithm with recombination.

This high density is, of course, bad for coverage path planning and yields tours with much redundant coverage. Close points on the boundary are frequently created by concave corners in the room, around which a circular robot has to make a circular turn, approximated by polygon with many segments. There are two steps we should perform, as shown in Figure 6.31: First, we can simplify the polygon and remove close points, e.g., by the usage of the Douglas-Peucker algorithm [170]. Because the connecting lines at the boundary can now intersect the unreachable boundary area, we have to move the points slightly inwards. Second, we

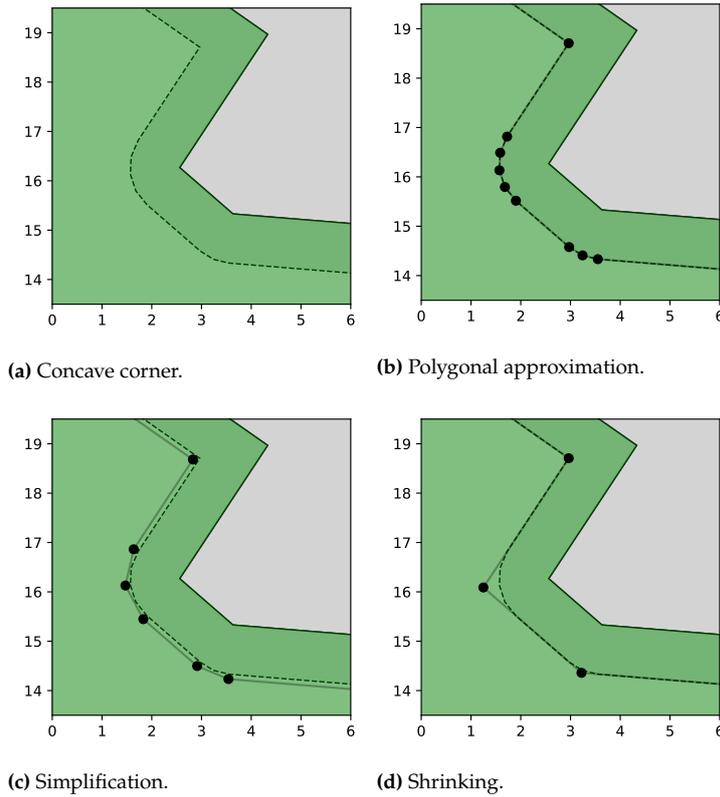


Figure 6.31: Concave corners in the area result in circular curves, under the assumption of a circular robot. Approximating such a curve results in many close boundary points that are problematic for many mesh generation methods. To circumvent this problem, we should simplify the resulting boundary by using, e.g., the Douglas-Peucker algorithm [170]. Because this can result in intersecting the boundary, the points should be moved slightly inwards. Additionally, we can work on a shrank boundary polygon instead of the feasible area (which potentially leaves a small part of the area uncovered).

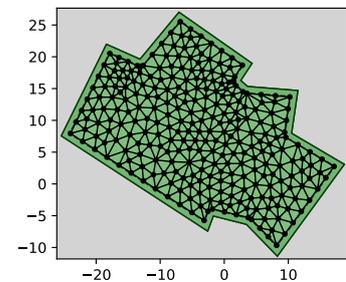
can simply shrink the boundary by the robot radius, instead of computing the feasible area by a Minkowsky sum. The resulting coverage path may miss some small area at this corner, but the underlying mesh allows a much better tour. We still need to perform a simplification for complicated boundary parts not created by curve approximations.

One does not need smooth meshes for all applications, and therefore many mesh generators only yield very rough grids out of the box. In combination with smoothing and optimization methods, these generators can still provide us with smooth meshes, see Figure 6.32.

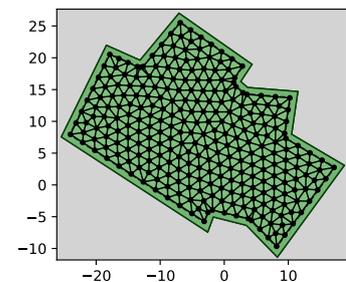
The classical algorithm of Du et al. [171] tries to achieve a centroidal Voronoi tessellation (CVT) by moving the points to the centroid of their Voronoi-cells (Lloyd's method [172]). Instead of the Voronoi diagram, one can also work on the dual Delaunay graph, as proposed by Chen and Holst [173]. The *optimesh*-library [174] implements these algorithms as well as some variants. The corresponding GitHub-page (<https://github.com/nschloe/optimesh>) gives a great overview with animations and also some experimental analyzes. Based on this experimental analysis, verified on some instances, we use the implemented CVT-variant *cvt-full* for all triangular meshes with a tolerance of 1×10^{-5} and 1000 iterations.

To compare all mesh generators is beyond the scope of this thesis, therefore we focus on a small selection of seven algorithms that seemed promising based on documentation and samples:

- The *MeshAdapt* (Δ MA), *Frontal-Delaunay* (Δ FD), *Frontal-Delaunay for Quads* (Δ FDQ), and *Packing of Parallelograms* (Δ PP) methods of *gmsht* [169] for triangular meshes.



(a) Without smoothing.



(b) With smoothing.

Figure 6.32: Many mesh generators only yield good meshes after smoothing.

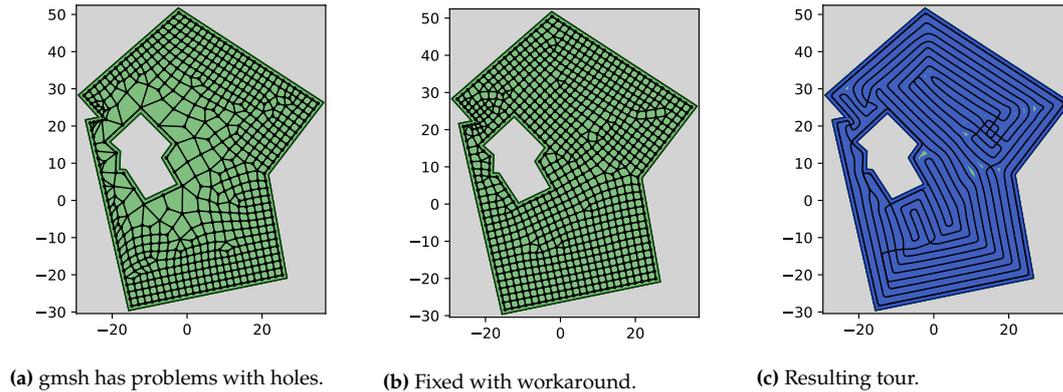


Figure 6.33.: The meshing algorithms of *gmsh* often have problems with holes in the polygon (at least using the *pygmsh* interface [177]). The boundary of the hole and the area surrounding it does not get populated enough. This can be fixed by enforcing fixed points on the hole boundary by replacing long segments with shorter ones. Using this fix, we can compute a good tour despite some artefacts in the mesh. This example uses *Packing of Parallelograms* with recombination.

- ▶ The *Frontal-Delaunay for Quads* (\square FDQ), and *Packing of Parallelograms* (\square PP) methods with recombination of *gmsh* [169] for quadrilateral meshes.
- ▶ The *dms*h-algorithm [175] that is inspired by *distmesh* [176].

To apply the same conditions to all algorithms, we use the same polygon shrinking and simplification for all mesh generators, even if they can handle complex, curved boundaries reasonably well, such as *dms*h.

The algorithms of *gmsh* often have problems with holes, see Figure 6.33. To fix this problem, we replace long segments in holes with shorter ones that approximate the desired edge length. If p_0p_1 is a long segment and d is the desired edge length with $d < \|p_0 - p_1\|$, we replace p_0p_1 with $\approx \|p_0 - p_1\|/d + 1$ equal sub-segments.

We compare the meshes on the same full coverage instances as the regular grids (Figure 6.24). The focus on full coverage allows us to focus more attention onto the coverage quality and the density of the meshes. The plots in Figure 6.34 show the average touring costs and coverage of the different meshes. The touring cost is measured in how much more expensive a tour is compared to the least expensive tour we found for this instance. A value of 20 % would indicate that the mesh yields tours that are on average 20 % more expensive than the least expensive tour. The coverage is measured regarding the whole area (note that not all of the area is always reachable). A value of, e.g., 95 % would indicate that the mesh yields tours that cover on average 95 % of the whole area.

Area is left uncovered not only due to turns and boundaries (as for regular grids) but also because the meshes do not always match the desired distances and can be too sparse in some areas. This especially happens for *dms*h and *Packing of Parallelograms*, thus, we also reduced their mesh size by 90 % and 95 % to increase the coverage for these cases. *dms*h failed to create a grid in 5 of the 200 instances because the internal geometric routines threw exceptions, likely due to numerical issues. These results are therefore ignored in analysis. One additional instance resulted in some disconnected areas after the polygon processing and is likewise ignored for all meshes.

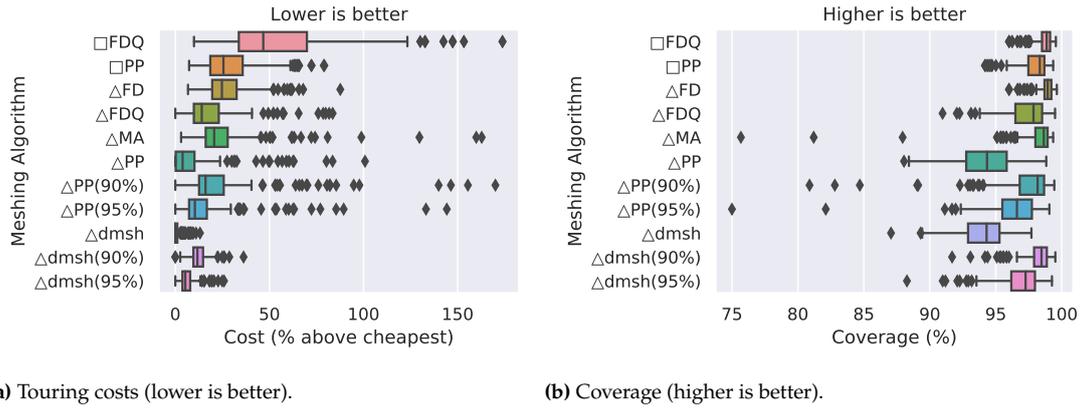


Figure 6.34.: Cost and coverage of tours in various meshes. The \square and \triangle indicate quadrilateral resp. triangular meshes. We can see low costs for *dmsb* and triangular *Packing of Parallelograms*, and a good coverage for the *Frontal-Delaunay*.

The *dmsb* and *Packing of Parallelograms*-algorithms yield the least expensive tours on average, with *dmsb* having an advantage for higher turn cost ratios. However, when taking a closer look into the data, most meshes can achieve good grids for specific instances. For example, relatively rectangular instances work very well with the quadrilateral mesh of *Packing of Parallelograms*. The coverage quality is nearly inverse to the touring costs. The triangular *Frontal-Delaunay* mesh performs best in coverage, and is on average around 25 % more expensive than the least expensive tour (that potentially has a lower coverage). Therefore, reducing the desired edge length for *dmsb* and *Packing of Parallelograms* results in better coverage at a small cost increase.

The mean runtime, as shown in Table 6.2, for some meshes can differ quite a lot with a mean runtime of 146.2 s for \triangle dmsb(90%) and only 89.5 s for \square PP. Interestingly, the quadrilateral meshes are not necessarily faster despite much smaller auxiliary graphs. This is likely due to the overhead induced by inefficient implementations of auxiliary steps, while the already optimized steps of the linear program and the matching algorithm are expected to be faster. Meshes created by *dmsb* are relatively slow but, contrary to the *gmsb*-algorithms, it is written in pure Python and, thus, not optimized for runtime. Otherwise, the triangular *Packing of Parallelograms* meshes are relatively fast. Note that the runtime refers to the whole solution process and not just the grid generation. The square grids are usually generated within one second and for the triangular grids, the optimization process of *optimesh* (in Python with NumPy and SciPy) can take up to a few seconds.

When looking onto which instances have been solved well on which mesh in Figures 6.35 and 6.36, no clear pattern is visible. The only observations that can be made are that *MeshAdapt* performs well on instances with many or larger holes, and the *Frontal-Delaunay* seems only to perform well on simpler instances. In the next sections, we will use *dmsb* with a 95 % point distance. In case of numerical issues, we will fall back on *Packing of Parallelograms*, also with 95 % point distance. The coverage of these meshes is on average sufficiently high with 96.9 % resp. 96.3 %.

Mesh	Runtime (s)
\square FDQ	129.7
\square PP	89.5
\triangle FD	137.6
\triangle FDQ	113.1
\triangle MA	120.7
\triangle PP	94.1
\triangle PP(90%)	127.4
\triangle PP(95%)	106.6
\triangle dmsb	114.6
\triangle dmsb(90%)	146.2
\triangle dmsb(95%)	123.3

Table 6.2.: Mean runtime with different meshes.



Figure 6.35.: Only *Packing of Parallelograms* yielded good quadrilateral meshes, shown here. The trajectory is displayed in black, the covered area in blue.

Future Work 6.8.

Some mesh generation algorithms, such as *distmesh* [176], allow to specify local distance functions that allow us to generate meshes with heterogeneous densities. Can this be used to create meshes for coverage path planning that are denser in important areas and lighter in less important areas?

Future Work 6.9.

The *centroidal Voronoi tessellation smoothing* [171] improves the triangular meshes significantly for our application. The quadrilateral meshes are currently not smoothed at all due to missing support in the used implementations. Can we smooth and improve the quadrilateral meshes? You can find more on quadrilateral meshes in [178] and [179].

6.5.3. Comparison

We now compare the performance of meshes and regular grids for partial and full coverage. For this, we generate instances as before based on the union of not-too-narrow quadrilaterals. We do the same for valuable and expensive areas. For the expensive areas, we combine all overlapping areas into a single area and do not allow overlap. Otherwise, we can easily get extremely high factors. The valuable areas are not multiplicative but additive and, thus, do not present this problem. The unsteadiness of overlapping valuable areas can even make the instances more realistic. Finding the right set of parameters such that we get well-balanced instances is difficult. Using purely random parameters, one often ends up with full coverage because the values are too high, or no coverage because the touring costs are too high. Thus, a careful fine-tuning of parameters is necessary to obtain diverse but interesting instances. See Figure 6.39 for a set of example instances with tours.

We evaluate the quality of the tours based on the objective, as described in Equation 6.1, which combines touring costs and coverage value. To make the objectives comparable over all instances, we divide every objective by the minimal objective known for the corresponding instance. We use the *dms*-algorithm with 95% edge length as representative of mesh-based coverage, and a regular triangular grid with line-based distances for grid-based coverage. To directly show the advantage of the partial-

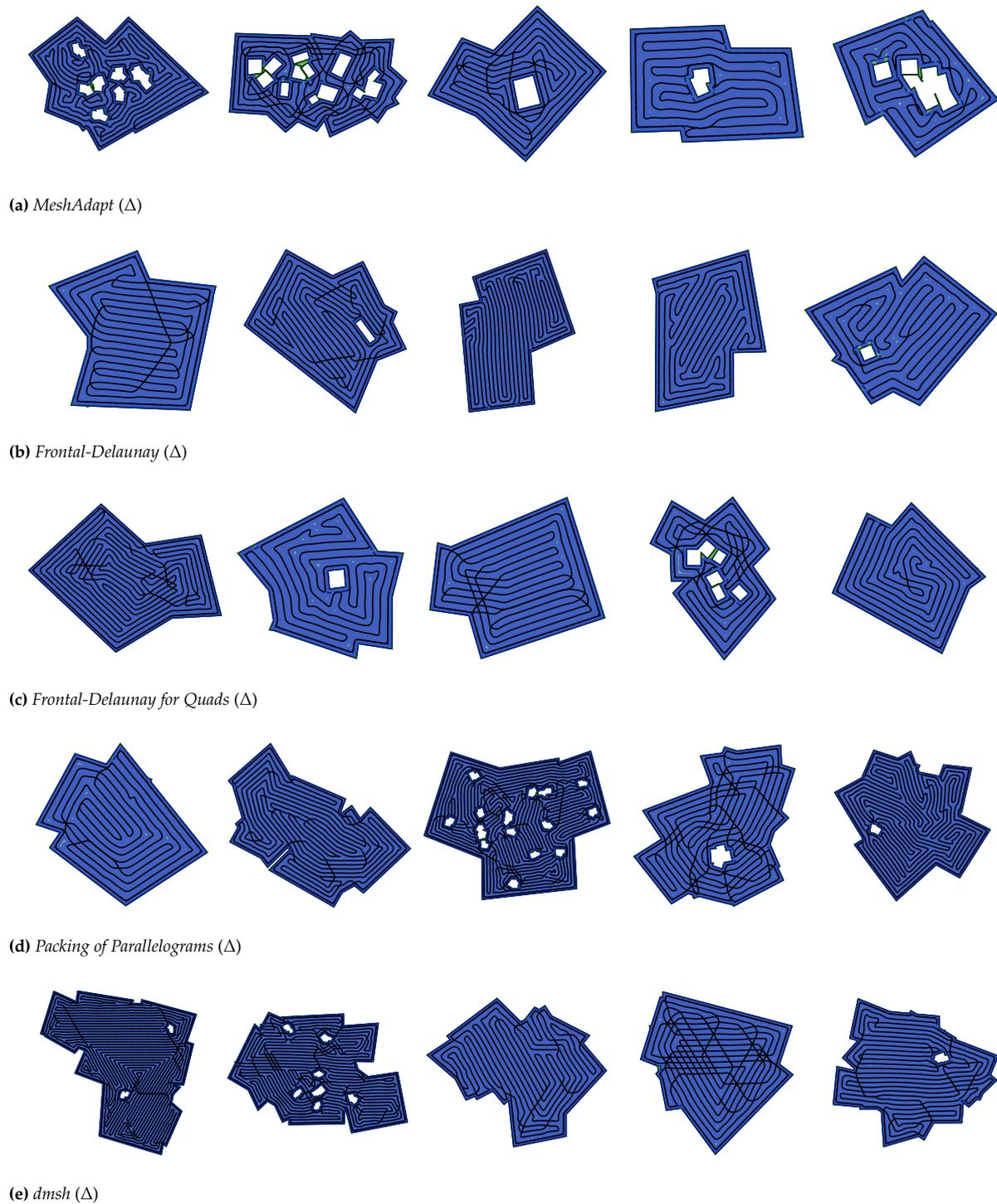
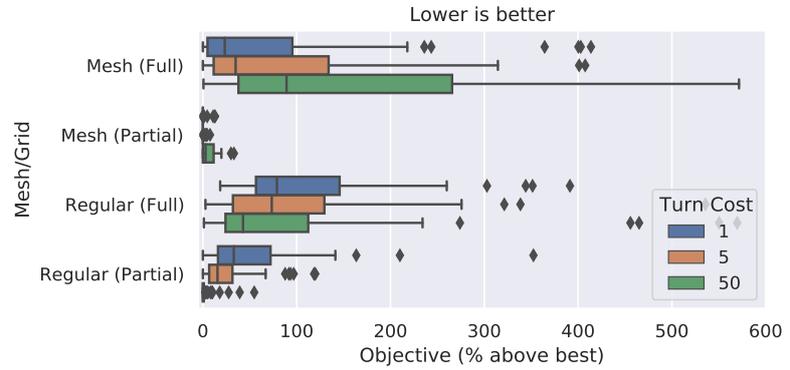


Figure 6.36.: Instances and tours with at least 95% coverage on which the corresponding mesh performed well (within 5% of the best). *MeshAdapt* performs well on instances with large hole areas, while the two *Frontal-Delaunay* versions perform primarily well for rather simple instances. By reducing the edge length in *dmsh*, we improve the coverage while only slightly increasing the costs. We consider *dmsh* with 95% edge length as our favourite, as it achieves a good performance and a sufficient coverage on average. If a more thorough coverage is necessary, we recommend simply to reduce the edge length of *dmsh*.

Figure 6.37.: Relative objectives for partial coverage computed with meshes and regular grids. Additionally, we compare partial with full coverage. The results are split for different turning cost weights, with 50 having a higher focus on turns than the traveled distance. We can see that the ability of partial coverage has a significant advantage. The pure touring costs of the partial-coverage tours on the mesh as well as the regular grid are similar, but the coverage on the mesh is much better.

coverage technique, we also compare the approaches with enforced full coverage.



In Figure 6.37, we can see that

- ▶ The partial mesh approach achieves the best objectives by a clear margin, independent of the weight of the turn costs. A deeper look into the data shows that the touring costs of the mesh are slightly worse than of the regular grid but this method covers much more valuable area.
- ▶ The performance of the meshes drops for higher turn costs, while the performance of the regular grids improves.
- ▶ The full-coverage tours are much more expensive than the partial-coverage tours. Using only partial-coverage for the evaluated instances provides a significant advantage. Note that some drastic outliers have been removed from the plot. They can be explained by small valuable areas that can be covered by a small tour which is obviously better than a full-coverage tour. However, many instances have larger valuable areas, and even the partial tours often cover much of the area.

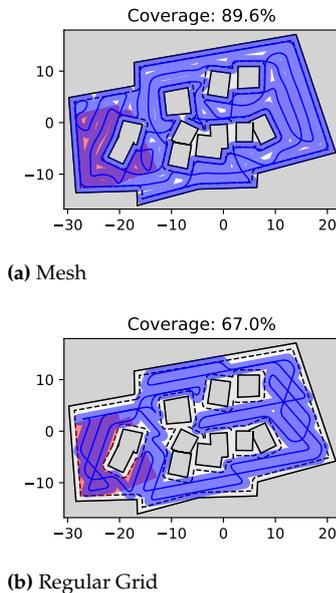


Figure 6.38.: An example for a difficult instance for which a mesh still achieves a reasonable coverage but a regular grid does not. A regular grid is too rigid to fit into the narrow passages.

When considering only the touring costs for full coverage, regular grids show to yield less expensive tours but with a lower coverage, as can be seen in Figure 6.40. Even when using point-based edge lengths for the regular grids, the coverage cannot compete with meshes while the costs are even higher. The significantly worse coverages often appear in complicated, small instances like in Figure 6.38. A regular grid cannot fit to the complex and narrow environment while a mesh can. Such instances also result in much higher touring costs for meshes because they do not skip the difficult areas. However, the meshes also leave more area uncovered than in less narrow areas. Because the mesh only tries to fit an average distance, the density in narrow areas can become too low, resulting in gaps.

For the further experiments, we focus on meshes because they achieve a more reliable coverage even for difficult instances.

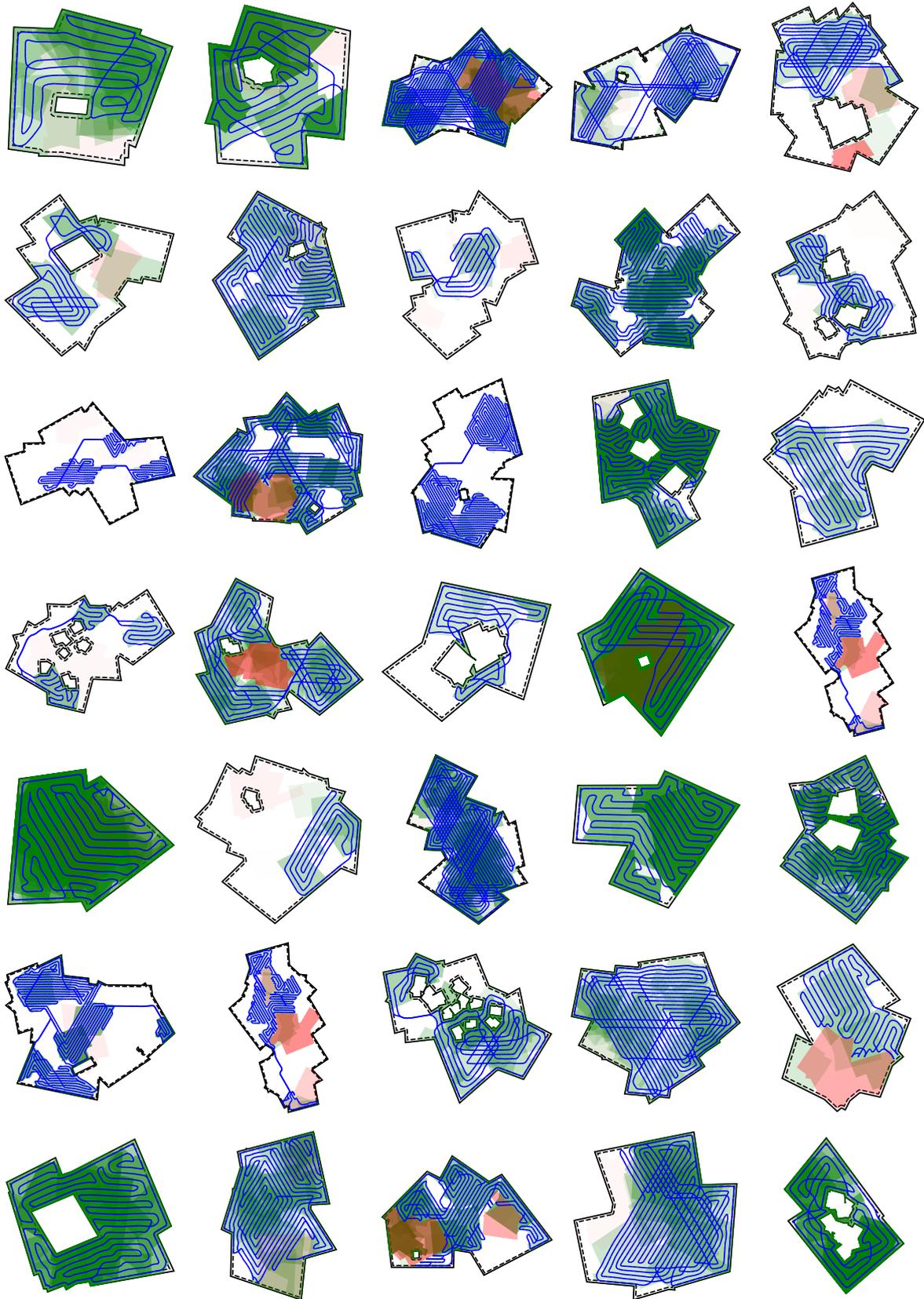
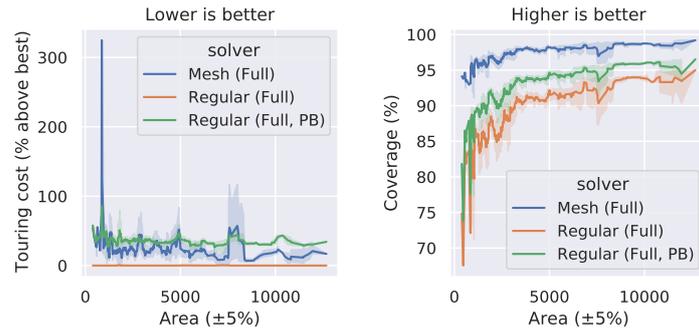


Figure 6.39.: Examples of instances and solutions for partial coverage. The trajectories of the tours are shown in blue and computed on a mesh. The green areas show valuable areas, and the red areas show expensive areas that multiply the touring costs inside. The intensity of the color indicates the value (a darker color has a higher value). Many solutions are good but show room for improvement, mostly due to a badly-aligned grid. However, some tours are better than anticipated based on the underlying objective, which is sometimes difficult to estimate at first glance.

Figure 6.40.: Meshes often yield slightly more expensive tours for full coverage, but also achieve a much better coverage. When using point-based (PB) edge lengths, the regular grid performs worse for full and partial coverage. The coverage generally increases for larger instances. The outliers are often small and complicated instances that a regular grid cannot cover properly but a mesh can. This results in high touring costs for the mesh and a low coverage for the regular grid.



6.6. Evaluation

Now that we know how to obtain a reasonably good graph representation of the instance, we evaluate the general performance of the algorithm and the influence of the optimization parameters. First, we look into the influence of the integralization optimization and the local optimization. Afterward, we check how good the solution is on the graph representation. For evaluating the quality in the polygon instance, independent of the chosen grid, we do not have any proper bound to compare it to. Instead, we compare how close the objective values in the graph and in the polygon are. Finally, we discuss the runtime of the implementation and possibilities to improve it.

There are three important metrics used in our evaluation:

P-Obj.: Refers to the objective value of the tour in the original polygonal instance, consisting of touring costs and opportunity loss for missed area.

G-Obj.: Refers to the objective value of the tour in the discrete graph representation, consisting of touring costs and the opportunity loss for missed vertices.

Fractional solution: Refers to the fractional solution on the discrete graph instance. It is a lower bound for the G-Obj., i.e., on the best tour we could obtain in the graph instance. It is not a lower bound for the P-Obj., because it is mesh-dependent and a different mesh may be able to yield a better solution.

We are interested in two properties:

- ▶ How good is our G-Obj., i.e., how much above the fractional solution is it? This shows us how well we solve the graph representation of the problem.
- ▶ How well does the G-Obj. approximate the P-Obj., i.e., how much above the G-Obj. is the P-Obj.? They only differ in the coverage value because, as we have seen in the previous section, the tour misses some coverage at turns and some areas are actually unreachable. For full coverage, P-Obj. and G-Obj. are identical.

To make the experiments more reliable, we expanded the previous instance set by 300 more instances, resulting in 500 instances with coverage value. These additional instances are of low- and medium-value density (coverable value divided by area). The previous 200 instances also

contain instances with very high value density that effectively enforce a full coverage.

6.6.1. Integralization

Let us first take a look on how much influence the branch-and-bound-based optimization of the fractional solution, computed in Section 6.4.2 (Fractional Solution) on page 117, has on the solution quality. The fractional solution helps us to guess the best passing direction, which induces high turn costs when guessed wrong. A more integral solution gives us a clearer picture of how the optimal solution is likely to look, and therefore helps us make more informed decisions resulting in better solutions. Note that a completely integral solution would allow us to skip directly to connecting the cycles, but this is highly unlikely to occur for non-trivial instances due to the NP-hardness of the problem.

To evaluate the usefulness of this optimization, we compute solutions with performing 0, 10, 25, 50, 100, and 200 branching steps. We always work on the leaf with the lowest objective value in the branch-and-bound tree, as this represents the current lower bound. This implies that the expected depth of the branch-and-bound tree is logarithmic because deeper leaves usually have a higher value.

The results are shown in Figure 6.41. In the first plot we can see that the difference of the G-Obj to the lower bound provided by the fractional solution decreases. 10 steps already make a visible difference. However, if we look at the second plot, we notice that the objective value of the final solution has barely changed. There is some fluctuation because some decisions in the solution process change, but the result is rather random and without a clear advantage. The improvement in the first plot can be understood if we look at the last plot. Here we see that the fractional solution does indeed improve. The increase of 1 % for 200 steps may not seem like much, but it closes the optimality gap by around 10 %. Interestingly, this does not help us to make better decisions or obtain better solutions, but it only proves that the computed solutions in the graph are closer to the optimum than indicated by the lower bound. In conclusion, this optimization indeed improves the fractional solution, but the unoptimized fractional solution is sufficient to select good atomic strips. As long as the proved quality factor is not of great importance, we can skip this optimization.

6.6.2. Local Optimization

The second optimization relaxes small areas of the solutions and recomputes the optimal solutions within these small areas using mixed integer programming. This is performed for cycle covers in Section 6.4.5 (Local Optimization) on page 122 and tours in Section 6.4.7 (Local Optimization) on page 126. There are two important parameters for this optimization: the number of iterations and the size of the relaxed area. If the area equals the full instance, we directly compute an optimal solution, but this is usually too expensive. Only small instances with less than ~100 vertices can be solved very quickly using the current technology. While we can solve many instances with 1000 and more vertices, this takes a

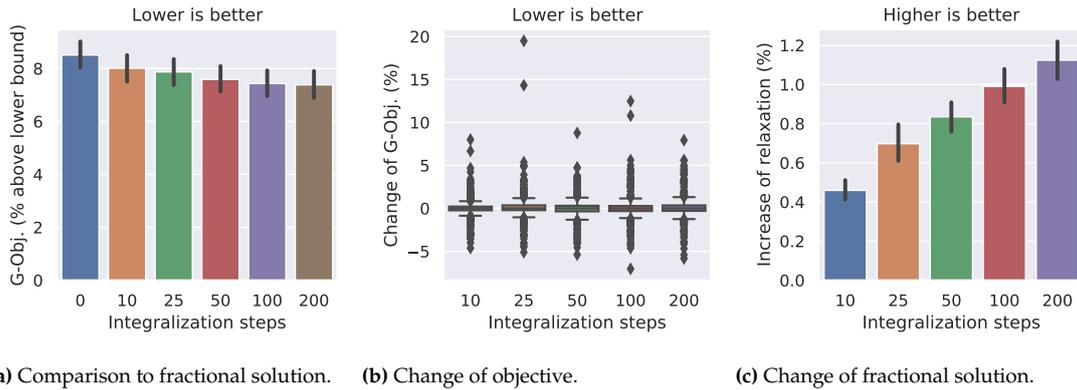


Figure 6.41.: Analysis of the improvement induced by optimizing the fractional solution via branch and bound steps. We can see that the gap between the solution and the fractional solution decreases, but the actual objective remains nearly the same. Only the lower bound improves but this does not help us finding a better solution.

minimum of several minutes. A large relaxed area can be substituted by more iterations of smaller areas to some degree. The runtime increases linearly with the number of iterations but exponentially with the size of the area. However, the NP-hard nature of the problem also implies that area cannot be fully substituted by iterations.

We again focus only on the graph representation of the instances, i.e., G-Obj., because the optimization focuses on the graph and does not know anything about the real instance. The important questions are:

- ▶ How much can we improve the solution using this optimization?
- ▶ Should we focus on optimizing the cycle covers or the tours? (While the tours are the final result, the cycle covers are less expensive to optimize.)
- ▶ How much influence do the number of iterations and the size of the area have?

To answer these questions, we computed solutions that performed a local optimization on either cycle cover or tour with 0, 10, 25, 50, 100 and 200 iterations and an area of 50 vertices. Additionally, we computed solutions that performed 50 iterations of the local optimization on either cycle cover or tour, but with a varying area of 0, 10, 25, 50, 75 and 100 vertices.

The results in Figure 6.42a show that the optimizations with an area size of 50 vertices yield a visible improvement for partial coverage in both steps. 10 iterations on the cycle cover already reduce the optimality gap (in comparison to the lower bound) by around 10%. The further iterations lose effectiveness, as can be expected because we prioritize the expensive areas, but an improvement remains visible. While the optimization is successful on cycle covers, it is even more impressive on tours. Here, the first 10 iterations lower the optimality gap by more than 20%. The further iterations also remain stronger than for cycle cover, but their improvements still decline quickly. This implies that the cycle covers are already nearly optimal, but the connection of the cycles to a tour is not very efficient. The local optimization on tours can easily find (locally) suboptimal parts in the connected solution and improve them visibly.

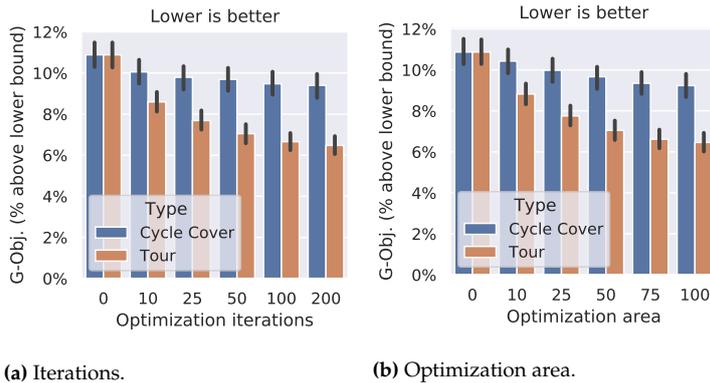


Figure 6.42.: Influence of the number of iterations and the optimization area for the local optimization depending on whether it is applied on cycle covers or tours. 10 % means that the objective in the graph-based instance is 10 % higher than the fractional solution (a lower bound on the achievable tour in this graph). Optimizing on tours is much more effective and few iterations and small optimization areas already have a visible effect.

The results in Figure 6.42b for varying area are surprisingly very similar: Doubling the area has a similar effect as quadrupling the iterations. One difference is that for optimizing cycle covers, the larger areas are more important than for tours. Optimizing only small areas with 10 vertices barely improves the solution. For tours, on the other hand, such small areas can already make a significant difference. This is very useful to know because optimizing 10 vertices is extremely fast and could still be done by brute-force. Thus, we can do many iterations with such small areas in a short time. Larger areas still have their advantage, and 50 iterations of size 100 are roughly as effective as 200 iterations of size 50.

The runtime differences for the number of iterations and the size of the area can be seen in Figure 6.44. Surprisingly, the runtime for larger areas looks nearly linear (caveat: the x-axis for iterations is exponential, but it is almost linear for area). However, this data should be used with caution because it is highly skewed. First of all, experiments were run in parallel on the same workstation, leading to a high variance. Second, the implementation is only optimized for quality but not for runtime. The connectivity detection, necessary to make sure that we did not accidentally disconnect the tour and have to insert constraints, is especially inefficient. Instead of only analysing the changed part, it always checks the whole solution with a procedure written in pure Python. This gives the tour variant a significant overhead which could be eliminated. The tour variant will still remain slower because the solution frequently gets disconnected and needs to be reconnected using additional constraints. These constraints become less efficient for larger areas because the solution develops more options to evade it. For larger optimization areas, additional constraints should be developed (compare with [27]).

For partial coverage, where the intermediate cycle cover can be very dispersed and difficult to connect, the optimization yields excellent results. Now we focus on full coverage: what does it look like when every vertex must be visited. The plot in Figure 6.43 shows that, also here, the tour optimization is more effective than the cycle cover optimization.

Interestingly, the iterations remain stronger than for partial coverage. A reason for this could be that the tours are on average longer and therefore have more area requiring optimization. Another explanation could be that the selection of the area is not as effective as for partial coverage.

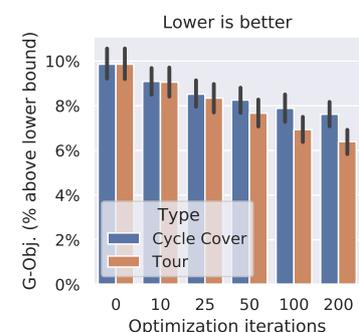
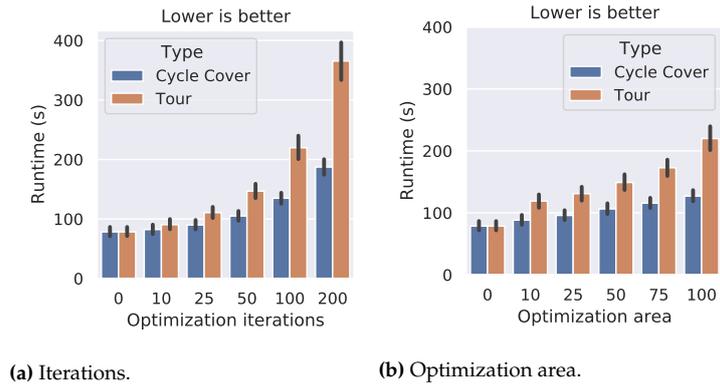


Figure 6.43.: Influence of the number of iterations for full coverage. The difference between performing the local optimization on cycle covers or tours is much smaller than for partial coverage.

Figure 6.44.: Runtime for more iterations or larger areas in the local optimization. Shown as the mean runtime in seconds over all 500 instances. Note that the connectivity detection necessary for tours uses only a naive implementation that takes a significant portion of the runtime for tours; this could be improved significantly by focusing only on the changes.



Note that the bars in the plot decline linearly but the iteration steps are doubled, implying an exponential decline.

Future Work 6.10.

Can we use the fact that small areas as well as a few iterations on larger areas already yield a significant improvement? We could first perform many fast iterations on small areas and then some slower iterations on larger areas. Alternatively, we can first try to optimize an expensive part by a small area and only increase the area if the small area is not sufficient.

6.6.3. Optimality Gap

Let us take a closer look at the solution quality in the graph instances. The plot in Figure 6.45 shows how the quality of the solution develops over the size of the instances.

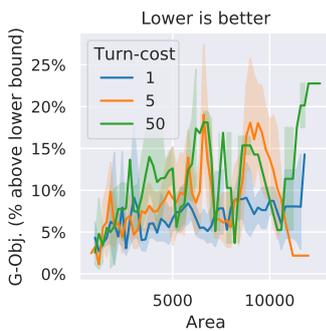


Figure 6.45.: Optimality gap (compared to fractional solution as lower bound) on graph instance over area size. It shows that the optimality gap increases for larger instances, and that lower weights on turn costs result in better solutions.

The quality is again measured by the difference of the objective value to the lower bound provided by the fractional solution. We can see that the objective is around 10% to 15% above the fractional solution, as we have already seen in the previous experiments. However, we make the new observation here that the quality slightly degrades for larger instances. Based on the tool radius of 1.0, the larger (graph) instances have a few thousand vertices. This degradation could be converging, but the data is relatively noisy and has too small a range to make any sure assumptions. The gap is generally smaller for lower turn costs, but this is not surprising because the turn costs make the problem combinatorially more complex. This influences at least the quality of the fractional solution, which provides us with the lower bound. Whether or not the actual solution has a larger optimality gap cannot be answered from the solution. Considering that the lower bound is probably off by a few percent, the overall solution quality can be called near optimal even for higher weights on turn costs.

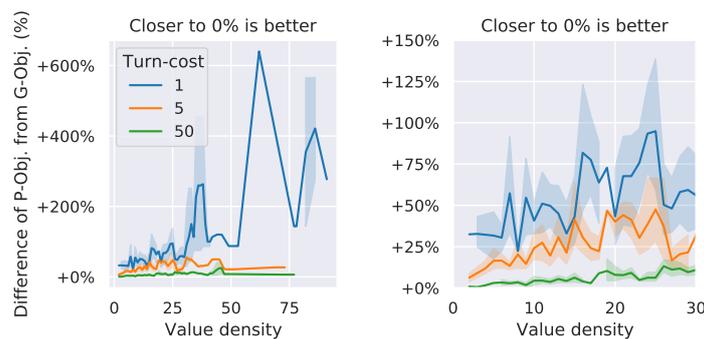
6.6.4. Objectives

Until now, we only considered the quality of the solution in the graph instance for the evaluation of the optimization. However, the graph

instance only approximates the polygon instance, and the objective on the original instance can differ from the objective on the graph instance. The culprit is the differing coverage: we have a lower coverage if we perform a turn than if we pass straight through a vertex. This problem can be overcome by using point-based grids instead of line-based grids, as discussed in Section 6.5 (Grids and Meshes) on page 128; but the touring cost would be much higher and we would have highly redundant coverages for straight passages. Thus, the P-Obj. will generally be worse, i.e., higher, than the G-Obj. due to unaccounted opportunity loss. In some cases, we will also have some valuable areas that are actually unreachable but are still assigned to the value of a vertex as it intersects its Voronoi-cell. The question is, how close is the P-Obj. to the G-Obj.?

To answer the question, we simply executed the default algorithm on all the 500 instances and calculated both objectives. We compared them using the percentage difference between the G-Obj. and the P-Obj., e.g., +50% means that the P-Obj. is 1.5× the G-Obj. While the P-Obj. can theoretically be lower than the G-Obj. if the value of the vertices has been underestimated, this is unlikely in our approach and did not occur in our instances. An important property for the quality of the G-Obj. is the proportion of the coverage value in the objective. A good indicator for this is the average value density in the feasible area. Some instances reached very high value densities by chance, and this resulted in the coverage value dominating the objective and forcing a full coverage. These instances are of course not representative.

The plots in Figure 6.46 show that for low densities and high turn costs, the P-Obj. is only minimally higher than the G-Obj., implying a good approximation of the polygon instance by the graph instance. This can be explained by two things: First, higher turn costs reduce the number of turns, meaning we lose some coverage, and increase the number of redundant passages. Second, the touring costs are higher due to the higher turn costs, meaning the relative value density is, thus, lower. For higher densities, the P-Obj. can be 600% above the G-Obj. because of missed coverage. In these cases, the G-Obj. is clearly not approximative.



(a) All value densities.

(b) Low and medium value densities.

Figure 6.46.: Difference between the original objective P-Obj. and the actually optimized objective G-Obj. For high weights on turn costs, the P-Obj. is only a few percent above the G-Obj. For high value densities and low weights on turn costs, the two objectives can drift apart drastically. Note that the higher value densities enforce full coverage as the coverage value dominates the objective and our solver is not optimized for 100% coverage.

To highlight the influence of the proportion of coverage value and touring costs in the objective, we can plot the difference in relation to the touring costs divided by the sum of all values in Figure 6.47. We see that extremes only occur if the touring costs are negligibly small compared to the coverage value. The instances with higher turn costs remain more accurate even when normalized in this way. Due to the high coverage

value for some instances, we may wrongly conclude that the previous experiments actually primarily compared the coverage quality and not the touring quality. First, the previous experiments were based on the G-Obj. which is much more generous regarding coverage. Second, only the missed coverage is represented by the objective value. These two points combined, result in a balanced focus on touring costs and coverage value. This is also visible in Figure 6.48.

Future Work 6.11.

Can we detect the proportion of the coverage value in advance and automatically place a stronger focus on coverage? The simplest approach is simply to swap from line-based grids to point-based grids if the value density is above some threshold that depends on the touring cost weights. A related but more advanced future work was mentioned in the previous chapter on Page 138 regarding dynamic mesh densities.

Figure 6.47.: As in Figure 6.46, but here we use the touring cost divided by the sum of values in the polygon instance as the y-axis. This neutralizes the effect of the turn cost weight on the relative value density. The advantage for instances with higher turn costs remains.

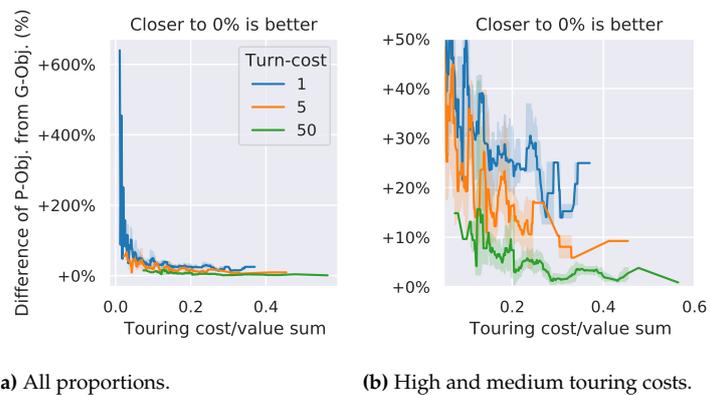
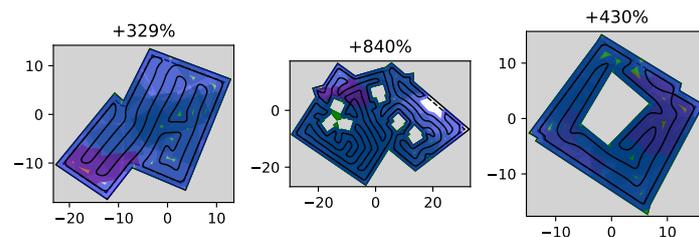


Figure 6.48.: Examples for instances with a huge difference between G-Obj. and P-Obj. The reason is solely in the high coverage value not because the tour is bad. These extreme examples also tend to be very small instances in which the random instance generator put by chance many high valued areas.



6.6.5. Runtime

In the previous chapter, we optimized the underlying algorithm to solve instances with over 300 000 vertices in regular square grids. In this chapter, we focused on generalization to make the algorithm applicable to real-world instances and to improve solution quality. The considered instances are much smaller and usually only have a few thousand or only hundred vertices, which can still result in complicated tours as seen in Figure 6.39. The runtime (a few minutes) of the used implementation,

as shown in Figure 6.49, can be too slow for real-life applications, e.g., a robotic vacuum cleaner. However, there are lots of opportunities for tuning:

- ▶ The used meshing algorithm *dms* is a slow, pure Python implementation. The implementations in *gms* are much faster and have only a slightly worse quality than *dms*.
- ▶ There are many expensive assertions: generally, checking for feasibility and connectivity takes a lot of time. Using optimized native data structures and code for these could reduce the runtime visibly (especially the tour-based optimization).
- ▶ Computing the connection costs between cycles and connecting them is time intensive, despite having a runtime of $O(n \log n)$. The many indirections in the look-up tables induced by Python can easily multiply the runtime by a factor of 100. Native data structures and code could also greatly decrease the runtime.

Future Work 6.12.

How much can we improve the runtime by replacing the Python-code by native C++-code? Remember that we were able to optimize regular square grids with up to 300 000 vertices in the previous chapter, using essentially the same algorithm.

Due to the generalization, it is unlikely that we can solve as large instances as in the previous chapter using code-optimizations alone. The implementation used in the previous chapter was able to use much faster integer arithmetic⁴, and also the instances had very a different character. To solve larger instances, we have to reduce the runtime of the Matching algorithm and the linear program. For the matching part, we already mentioned in the corresponding chapter that we could try to reduce the number of edges in the auxiliary problem. Most of the edges are actually not necessary, and additionally, thanks to the primal dual implementation, we can add edges later without restarting the algorithm if we notice some important edges are missing. Another more general approach for both parts is to partition the area and solve much smaller subproblems. At that point, we only want a cycle cover, meaning connectivity between the subproblems is not necessary.

Future Work 6.13.

Can we partition very large instances into smaller instances in order to compute cycle covers faster? This is very similar to our local optimization but on a larger level. Such an approach would be easy to parallelize and scale for quite large instances.

This also opens up the following question, which can be much easier implemented and evaluated:

Future Work 6.14.

How good are the solutions if we do not use the approximation algorithm to get a first solution, but instead directly use the local

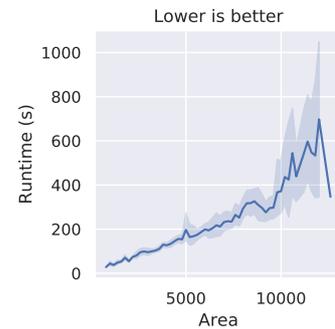


Figure 6.49.: Mean runtime over instance size of the Python-implementation.

4: Less because of the actual arithmetic operations but more due to numerical issues within the algorithms themselves.

optimization, i.e., only perform Steps 1, 5, 6, and 7? This approach has less of a chance to detect the global structure of the instance and will probably yield worse tours, but has the redeeming factor of potentially being much faster.

6.7. Conclusion

Throughout this chapter, we first generalized the approximation algorithm of the previous chapter to work on polygonal instances, and we also introduced some optimizations. Afterward, we discussed how to get a good graph representation of the polygonal environment using regular grids and meshes. At the end, we evaluated the algorithm and saw that we were able to compute solutions that are on average close to optimum (10%) on the graph representation. The graph representation, on the other hand, can be problematic, and the objectives in both representations can diverge because of missed coverage, but for higher turn costs and lower value densities, the graph representation is very accurate. Overall, the approach shows to be very effective when turn costs are important and missing around 5% of the coverage is not critical.

There is a lot of potential future work highlighted throughout the chapter, that can improve the performance on other kinds of instances as well as improving the runtime. The most critical and difficult part for computing better solutions is clearly the creation of good graph representations, i.e., grids and meshes. Here, significant improvements can still be achieved, while the runtime remains the primary concern for optimizing the resulting graph representation.

The approach can also be modified to many different problem variants. A common and more difficult problem variant is the presence of an energy or time budget. This can be approximated to some degree by scaling the touring costs when the tour is using too much or little energy, but a direct optimization is more challenging. Of course, this can take many iterations and may not even yield a satisfying tour. Alternatively, one can first compute a regular cycle cover and then select a subset of the cycles that fit the budget. However, one can also easily imagine an instance where this approach fails.

Future Work 6.15.

How can we optimize the problem of maximizing the covered value with a limited touring cost budget? Without turn costs, this equals the *Budget TSP* on the graph abstraction. With turn costs, the problem becomes more difficult and requires new algorithmic ideas.

Part III.

CAPTURING TRAJECTORIES

Probing a Set of Trajectories to Maximize Captured Information*

7.

In this chapter, we study a trajectory analysis problem we call the **TRAJECTORY CAPTURE PROBLEM (TCP)**: for a given input set \mathcal{T} of trajectories in the plane, and an integer $k \geq 2$, we seek to compute a set of k points (“portals”) to maximize the total weight of all subtrajectories of \mathcal{T} between pairs of portals. This problem naturally arises in trajectory analysis and summarization.

We show that the TCP is NP-hard (even in very special cases), and we give some first approximation results. Our main focus is on attacking the TCP with practical algorithm engineering approaches, including integer programming (to solve instances to provable optimality) and local search methods. We study the integrality gap arising from such approaches. We then analyze our methods using different classes of data, including benchmark instances that we generate. Our goal is to understand the best-performing heuristics, based on both solution time and solution quality. We demonstrate that we are able to compute provably optimal solutions for real-world instances.

7.1 Introduction	153
7.1.1 Overview	154
7.1.2 Related Work	155
7.2 Preliminaries	156
7.3 Analytical Results	156
7.3.1 Complexity	157
7.3.2 Approximation	160
7.4 Algorithm Engineering .	161
7.4.1 Integer Programming . .	161
7.4.2 Heuristics	164
7.4.3 Generating Benchmark Instances	165
7.4.4 Experimental Evaluation	165
7.5 Conclusion	170

7.1. Introduction

In recent years, the progress in technical capabilities has resulted in massive amounts of trajectory data for cars, trucks, trains, aircraft, ships, people, and animals being collected at increasing rates. This presents major challenges for storing and evaluating this ever-growing data, as well as for extracting useful information; this motivates the search for data structures and algorithms that capture some of the most important and useful aspects of such trajectories. At the same time, the availability of large volumes of data makes it possible to consider useful aspects that were previously unavailable due to the lack of data or algorithmic evaluation methods, such as collecting useful information along the traveled trajectories.

One such means of analyzing a set \mathcal{T} of trajectories is to determine “popular pairs” (p_1, p_2) of locations (or location/time pairs) for which there is a significant “value” of the trajectories \mathcal{T} going between those points. This value can arise from aggregated data between checkpoints, such as the total passenger-distance or the accumulated total pollution along the way; it also comes up through the use of tomographic methods (i.e., determining physical phenomena by measuring aggregated effects along the path between two sensors), which are highly important in the context of many other application areas, such as in astrophysics [29]. A limiting factor is usually the need to pick a set of locations of finite cardinality, i.e., placing a limited number of toll booths, cameras for average speed measurement, various other types of sensors, or abstract

* The content of this chapter was presented at SEA 2020 [6]. Many thanks to the coauthors Sándor Fekete, Alexander Hill, Tyler Mayer, Joseph Mitchell, Ojas Parekh, and Cynthia Phillips for the fruitful collaboration.

collections of focus points for sampling trajectories. For an example, consider the scenario shown in Figure 7.1, which corresponds to more than 500 000 data points that arise from the trajectories of over 250 taxi cabs in San Francisco. Our goal is to identify a small subset of locations that allow us to capture as much of the movement information, in terms of weighted distance between checkpoints, as possible.

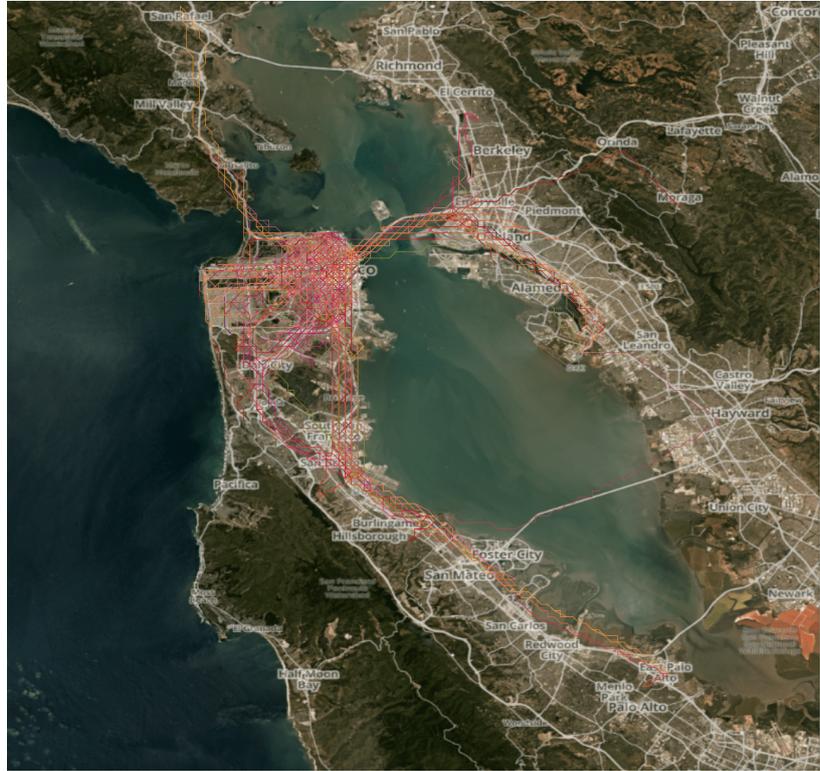


Figure 7.1: A set of taxi trajectories after preprocessing in San Francisco Bay Area. (Satellite images courtesy of Planet Labs Inc.)

In this chapter, we study the **TRAJECTORY CAPTURING PROBLEM (TCP)**, a generalization of “popular pair” computation: Given a set \mathcal{T} of trajectories and an integer $k \geq 2$, determine a set of k portals (points) to maximize the sum of the weights of the inter-portal subtrajectories in \mathcal{T} ; such subtrajectories are said to be “captured” by the set of portals. After establishing that the TCP is NP-hard, and giving some first approximation bounds, we focus on algorithm engineering methods for attacking the problem practically.

7.1.1. Overview

We provide results both for algorithmic theory and algorithm engineering aspects of the **TRAJECTORY CAPTURING PROBLEM**.

- ▶ We prove NP-hardness for the TCP, even for instances with trajectories consisting of individual axis-parallel segments.
- ▶ We establish two approximation algorithms. One has approximation factor K if the input trajectory set decomposes into K subsets where within each subset no two segments cross, though they can overlap, (K noncrossing subsets). The other has an approximation factor Δ , the maximum number of input trajectories hit by any single point.

- ▶ We develop an Integer Program (IP) to solve TCP instances to provable optimality.
- ▶ We show that, in general, the IP formulation has unbounded integrality gap*. For inputs decomposing into 2 noncrossing subsets (e.g., arising from axis-parallel segments), we show that the integrality gap is at most $\frac{k}{\lfloor k/2 \rfloor}$ (Theorem 7.4.2).
- ▶ We develop methods for generating challenging benchmark instances for experimentation. For geometric instances, based on segment arrangements, this requires care to address geometric robustness and accuracy.
- ▶ We compute provably optimal solutions for general instances up to thousands of candidate capture points.
- ▶ We give provably optimal solutions for even larger instances, with up to 7000 possible capture points, for instances based on axis-parallel segments, where we find the integrality gap to be quite small.
- ▶ Using the IP solutions as a reference, we perform a thorough computational study using heuristic algorithms (a greedy algorithm, iterated local search, simulated annealing, and an evolutionary algorithm), with various settings, to understand how heuristics perform on various instances, in terms of time and solution quality.
- ▶ We demonstrate our methods on real-world instances, including a provably optimal solution for taxi-cab data on 250 trajectories, with more than 500 000 individual geographic data points.

Given the broad range of potential applications, scenarios and assumptions, we do not claim (or even aim) to provide a final set of methods for the general problem. Instead, we focus on demonstrating that a number of modeling and optimization approaches can provide a promising range of insights and tools for future work from various directions.

7.1.2. Related Work

Related to our problem is the well-studied GEOMETRIC HITTING SET PROBLEM (GHS), in which one seeks a smallest cardinality set of points to hit a given set of lines, segments, or trajectories in the plane; as a single point suffices to capture all of a trajectory it lies on, achieving large objective values for the GHS is easier than for the TCP. The GHS is known to be NP-hard, hard to approximate (below a threshold), and some natural geometric cases have constant-factor approximation algorithms; see, e.g., [180], and the references therein.

There is a vast literature on problems of analyzing, clustering, mining, and summarizing a set of trajectories. For an extensive survey of trajectory data mining methods, see Zheng [181] and Zheng and Zhou [182]. Notions of “flocks” and “meetings” have been formalized and studied algorithmically [183–186]. Gudmundsson, van Kreveld, and Speckmann [187] define *leadership*, *convergence*, and *encounter* and provide exact and approximate algorithms to compute each. Andersson, Gudmundsson, Laube, and Wolle [188] show that several *Leader-Problem* (LP) variants

* The integrality gap is $\max_{\mathcal{F}} \frac{\text{LP}(\mathcal{F})}{\text{IP}(\mathcal{F})}$, where \mathcal{F} is a TCP instance, $\text{IP}(\mathcal{F})$ is the optimal IP solution value and $\text{LP}(\mathcal{F})$ is the optimal solution value to the linear programming (LP) relaxation.

(*LP-Report-All*, *LP-max-Length*, *LP-Max-Size*) are all polynomially solvable and provide exact algorithms. Buchin et al. [189] present a framework to fully categorize trajectory grouping structures (grouping, merging, splitting, and ending of groups). Other approaches to trajectory summarization naturally include cluster analysis, of which there is a large body of related work. Li, Han, and Yang [190] consider rectilinear trajectories and show how to cluster with bounding rectangles of a given size. Several approaches (e.g., [191–194]) consider density-based methods for clustering sub-trajectories. Lee, Han, Li, and Gonzalez [192] take it one step further by considering a two-level clustering hierarchy that first accounts for regional density and then considers lower-level movement patterns. Li, Ding, Han, and Kays [195] consider a problem (related to [187]) in which they seek to identify all *swarms* or groups of entities moving within an arbitrary shaped cluster for a certain, possibly disconnected, duration of time. Also, Uddin, Ravishankar, and Tsotras [196] consider finding what they call *regions of interest* in a trajectory database.

In motivating tomographic applications, the number of checkpoints is an important constraint in the use of discrete tomography, e.g., in astrophysics (Korth et al. [29]).

7.2. Preliminaries

We are given a set of trajectories \mathcal{T} , each specified by a sequence of points, e.g., in the Euclidean plane. We seek a set $P = \{p_1, \dots, p_k\}$ of k *portals*, i.e., selected points that lie on some of the trajectories. While our practical study focuses on instances in which the trajectories \mathcal{T} are purely spatial, e.g., given as polygonal chains or line segments in the plane, our methods apply equally well to more general portals and to trajectories that include a temporal component and live in space-time. More generally, we are given a graph \mathcal{G} , with length-weighted edges, and a set of paths within \mathcal{G} . We wish to determine a subset of k of the nodes of \mathcal{G} that maximizes the sum of the (weighted) lengths of the subpaths (of the input paths) that link consecutive portals along the input paths.

We seek to compute a P that maximizes the total *captured weight* of subtrajectories between pairs of portals. For a trajectory $\tau \in \mathcal{T}$, if there are two or more portals of P that lie along τ , say $\{p_{i_1}, \dots, p_{i_q}\}$ (for $q \geq 2$), then the subtrajectory, $\tau_{p_{i_1}, p_{i_q}}$, between p_{i_1} and p_{i_q} is *captured* by P , and we get credit for its weight $f(\tau_{p_{i_1}, p_{i_q}})$. (For many of our instances, $f(\tau_{p_{i_1}, p_{i_q}})$ corresponds to the Euclidean distances, denoted by $|\tau_{p_{i_1}, p_{i_q}}|$, but our methods generalize to other types of weights.) Let $f_P(\tau)$ denote the captured weight of trajectory τ by the portal set P . The **TRAJECTORY CAPTURE PROBLEM (TCP)** is then to compute, for given \mathcal{T} and k , a set of k portals $P = \{p_1, \dots, p_k\}$ to maximize $\sum_{\tau \in \mathcal{T}} f_P(\tau)$.

7.3. Analytical Results

The TCP is NP-hard and hard to approximate for general graphs even when all trajectories have weight 1. Given a graph, let each edge be a trajectory. Then an optimal solution to TCP gives the densest k -node

subgraph. Manurangsi [197] showed that, assuming the exponential time hypothesis, there cannot be an $n^{\frac{1}{\log \log n^c}}$ -approximation algorithm for the DENSEST k -SUBGRAPH problem for any constant $c > 0$.

In the following, we give more specific results for a range of geometric TCP versions.

7.3.1. Complexity

In the one-dimensional setting, the underlying graph \mathcal{G} is a path, and the input trajectories $\mathcal{T} = \{(a_1, b_1), \dots, (a_n, b_n)\}$ are a set of subpaths of \mathcal{G} , specified by pairs of integers, a_i, b_i . A solution to the TCP then consists of k points, $P = \{p_1, \dots, p_k\}$, w.l.o.g. indexed in sorted order, $p_1 < p_2 < \dots < p_k$.

Theorem 7.3.1 *The one-dimensional TCP can be solved exactly in polynomial time.*

Proof. Note that we can split any trajectory at a portal because the portal would immediately continue the capturing. This splits the instance in an independent left and right side. Further, only the endpoints $E = (a_1, b_1, a_2, b_2, \dots)$ of the trajectories are relevant positions for the portals. This allows us to devise a dynamic program: Let $V_i(x)$ denote the maximum possible length of \mathcal{T} captured by i portals (p_0, \dots, p_{i-1}) to the left of x with $p_{i-1} = x$.

$$V_i(x) = \begin{cases} 0 & \text{if } i = 1 \\ \max_{x' \in E, x' \leq x} \{V_{i-1}(x') + \underbrace{|\{s \in \mathcal{T} \mid x', x \in s\}| \cdot (x - x')}_{\text{captured by } x' \text{ and } x}\} & \text{else} \end{cases}$$

We now can compute the optimal solution in $O(n^2k)$ by $\max_{x \in E} V_k(x)$. \square

For two-dimensional instances, things get hard as the following two theorems show.

Theorem 7.3.2 *The TCP is NP-hard, even for an input \mathcal{T} of n line segments in the plane.*

Proof. The reduction is from the HITTING LINES problem: Decide if there exists a set of k points that hit every one of a given set of n lines in the plane.

Consider an instance, $\mathcal{L} = \{\ell_1, \dots, \ell_n\}$, of n lines in the plane. Let $\delta > 0$ be the radius of a disk, D_δ , centered at the origin $(0, 0)$, that is large enough to contain all intersection points (crossing points) in the arrangement of \mathcal{L} ; it suffices to select δ greater than the Euclidean distance from the origin to any crossing point, $\ell_i \cap \ell_j$. Let $R \ll \delta$ and consider the much larger disk, $D_{R+\delta}$, centered at the origin; it will suffice to pick $R = 2n\delta$. Each line ℓ_i intersects D_δ in a single segment, s_i , and intersects the annulus $D_{R+\delta} \setminus D_\delta$ in two segments, s'_i and s''_i , with s'_i in the halfspace $\{(x, y) \in \mathbb{R}^2 : y \leq 0\}$ (and s''_i in the halfspace with $y \geq 0$).

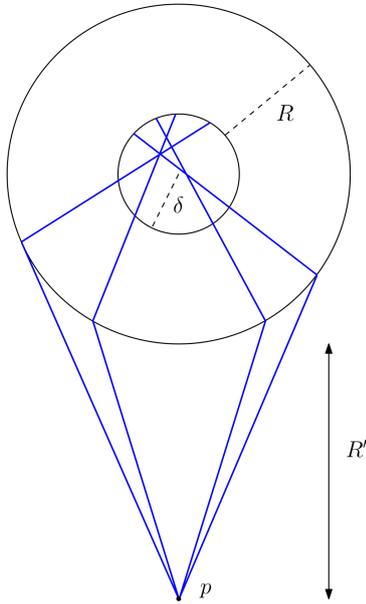


Figure 7.2.: Reduction from the problem HITTING LINES.

We consider the instance of TCP with the following $2n$ line segments as input: the n segments, $s_i \cup s'_i$, obtained by concatenating s_i and s'_i , and the n segments σ_i that connect the endpoint of s'_i (the one at distance $R + \delta$ from the origin) to a point p , at distance $R' + R \ll R$ from the origin; it will suffice to pick $R' = 2nR$. Refer to Figure 7.2.

We claim that there is a hitting set of size k for \mathcal{L} if and only if it is possible to capture at least length $\sum_i (|s'_i| + |\sigma_i|)$ using a budget of $n + 1 + k$ points in the instance of TCP.

First, if there exists a hitting set of size k for \mathcal{L} , then in the instance of TCP, we place k hit points (within D_δ) to hit all of the segments $s_i \cup s'_i$, and place n hit points at the endpoints of these n segments that lie on the boundary of $D_{R+\delta}$, and one point at p ; this suffices to capture at least length $\sum_i (|s'_i| + |\sigma_i|)$.

For the converse, the only way to capture length at least $\sum_i (|s'_i| + |\sigma_i|)$ is to place hit points at p , at the n points $\sigma_i \cap s'_i$, and at k points within D_δ that hit all n of the segments s'_i . (Lengths captured within D_δ are very small compared to the lengths captured outside of D_δ .) \square

The proof of Theorem 7.3.2 uses a construction involving segments of *many* orientations whose pairwise intersections may only be single points. The following shows that the TCP is already NP-hard for segments of *two* orientations, provided that two intersecting segments may be collinear, and three different segments can intersect in a single point.

Theorem 7.3.3 *The TCP is NP-hard, even for an input \mathcal{T} of n line segments of at most two orientations in the plane, with any two segments intersecting in at most a single point (there are no overlapping pairs of segments) but possible collinearity.*

Proof. Without loss of generality, we assume that the two orientations are horizontal and vertical; i.e., the segments of \mathcal{T} are axis-parallel.

The reduction is from 3-SAT, in which one is to decide if there is a truth assignment for n Boolean variables $x_i : 1 \leq i \leq n$ that satisfy a set of m clauses in conjunctive normal form. The construction is similar to that used to show NP-hardness of the hitting set problem on axis-parallel segments [180], with some key new features. See Figure 7.3 for the overall construction.

First, we place m vertical *clause* segments, each of length nm , evenly spaced at unit distance; these are shown in blue at the top of Figure 7.3. We also place $2n$ vertical *variable* segments of length nm (also shown in blue) at unit distance to the right and left of the clause segments, such that their top vertices line up with the bottom vertices of the clause segments; these variable segments are slightly shifted vertically and horizontally by $\Theta(\varepsilon)$ for a sufficiently small ε (it suffices to set $\varepsilon = \Theta(1/mn)$).

For each of the n variables, we add a set of 2 *literal* horizontal chains of (approximately) unit segments (shown in orange in the figure); each chain consists of $m + 1$ such segments, and two consecutive segments in the same chain intersect only in their shared endpoint; these are shown by red and green dots in the figure. Similarly, the first and last

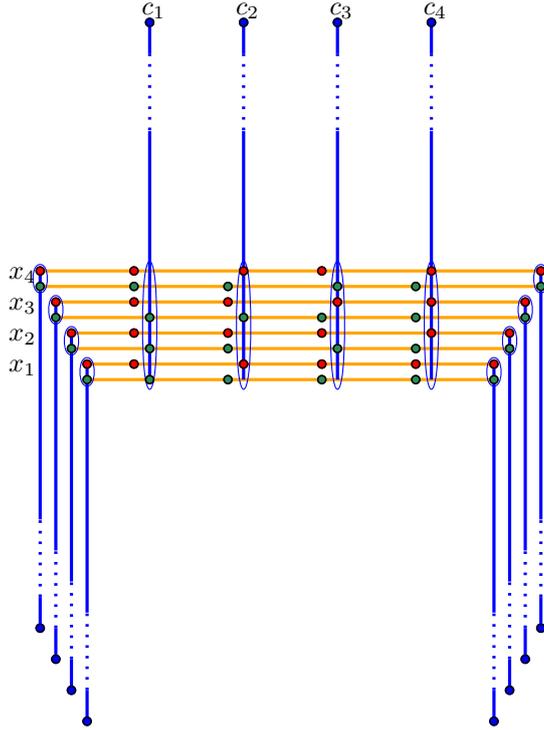


Figure 7.3.: Overview of the hardness construction, for the 3SAT instance $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_3 \vee \bar{x}_4) \wedge (x_2 \vee \bar{x}_3 \vee x_4) \wedge (\bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4)$. For an instance with m clauses and n variables, it consists of $2n + m$ vertical (blue) segments, each of length nm , and $2n(m + 1)$ horizontal (orange) segments of unit length (up to small modifications of size $O(\varepsilon)$). The red and green dots indicate points at which different segments meet. With $4n + m + nm$ portals, a total distance of length at least $n(m + 1) + 2n^2m + nm^2 - \Theta(nm\varepsilon)$ can be captured if and only if the 3SAT instance has a satisfying truth instance.

endpoint of such chains of horizontal segments lie on the corresponding left and right vertical variable segments. The vertical distance between the two chains is $\Theta(\varepsilon)$. For each variable, the lower chain corresponds to a true assignment (corresponding to green dots), while the upper chain corresponds to a false assignment of that variable (corresponding to red dots).

Finally, the lengths of the horizontal segments for a literal are adjusted by $\Theta(\varepsilon)$, so that precisely the dots corresponding to the literals in a particular clause are positioned on the vertical segments for that clause.

Now we claim: With $4n + m + nm$ portals, a total distance of length at least $n(m + 1) + 2n^2m + nm^2 - \frac{1}{2}$ can be captured, if and only if the 3SAT instance has a satisfying truth instance.

It is straightforward to see that a satisfying truth assignment induces a set of $4n + m + nm$ portals that capture a total distance of length at least $n(m + 1) + 2n^2m + nm^2 - \Theta(nm\varepsilon)$: Simply pick the upper endpoints of vertical clause segments and the bottom endpoints of vertical variable segments, along with all of the endpoints of horizontal literal segments for the chain corresponding to the truth assignments of a variable. This captures the full distance of $m + 1 - \Theta(n\varepsilon)$ of one horizontal chain of literal segments for each of the n variables, $nm - \Theta(n\varepsilon)$ of each of the $2n$ vertical variable segments, as well as $nm - \Theta(n\varepsilon)$ of each of the m vertical clause segments.

For the converse, first observe that any solution satisfying the bound must capture the full length of all vertical segments, up to a total difference of at most $\frac{1}{2}$. As a consequence, we can assume that $2n + m$ portals must be placed at endpoints of the vertical segments, as indicated by the $2n + m$ blue dots in the figure. Furthermore, each vertical segment must have a second portal near its other endpoint; without decreasing the value of

the solution by more than $\Theta(n\varepsilon)$, we can assume that each such portal is placed on one of the endpoints of a horizontal literal segment. This captures a total distance of $2n^2m + nm^2 - \Theta(nm\varepsilon)$ from the vertical segments.

On the other hand, this leaves at most $2n + nm = n(m + 2)$ (non-blue) portals to capture a distance of at least $n(m + 1) - \frac{1}{2}$ from the horizontal unit-length segments. If we consider the auxiliary graph in which these portals are represented by vertices, and two vertices are adjacent if they capture the same horizontal edge, we conclude that this graph decomposes into a set of paths; furthermore, a path consisting of $p + 1$ vertices and p edges captures a distance of at most p . As a consequence, there can be at most n such paths, otherwise we capture a horizontal distance of at most $n(m + 1) - 1$. Because the longest possible length of a path is $m + 1$, which occurs if and only if we pick all endpoints in a full horizontal literal chain, it follows that we must pick precisely n such full chains, with either of their two endpoints on the two vertical segments for the corresponding variable. As this leaves $2n$ portals to be positioned near the upper endpoints of the $2n$ vertical variable segments, and we need at least one portal on each of the vertical variable segments, we conclude that for each variable we must pick precisely one literal chain of endpoints, corresponding to a truth assignment. Finally, the choice of portals must also pick at least one portal near the bottom end of each vertical variable segment; this is only possible if the choice of literal chains produces at least one satisfying literal for each clause. This concludes the claim. \square

7.3.2. Approximation Algorithms

Consider first the case in which the set \mathcal{T} of input trajectories is the (disjoint) union of K subsets of trajectories, $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2 \cup \dots \cup \mathcal{T}_K$, with each subset \mathcal{T}_i having the following *path property*: For any connected component of the intersection graph of \mathcal{T}_i , the trajectories in that component are all subpaths of some path in the union of \mathcal{T}_i . This condition holds, for example, if \mathcal{T} is a set of line segments of K distinct orientations.

Theorem 7.3.4 *The TCP for an input set $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2 \cup \dots \cup \mathcal{T}_K$, with each subset \mathcal{T}_i having the path property, has a polynomial-time K -approximation algorithm.*

Proof. Within each class \mathcal{T}_i , the path property implies that the trajectories behave like intervals on a line, so our one-dimensional dynamic programming solution applies. Selecting the best solution that uses all k points for one of the \mathcal{T}_i gives a polynomial-time algorithm with approximation ratio K . \square

Theorem 7.3.5 *The TCP for an input set \mathcal{T} of arbitrarily overlapping/crossing trajectory paths in the plane having bounded depth Δ (i.e., no point of \mathbb{R}^2 lies in more than Δ input trajectories) has a polynomial-time Δ -approximation algorithm, for any even number, k , of portals. (If k is odd, the approximation factor is at most $\Delta(1 + \frac{1}{k-1})$.)*

Proof. Consider an optimal set, H^* , of k hit points, capturing total trajectory length L^* . For each hit point $h \in H^*$, which lies on $\delta \leq \Delta$ trajectories of \mathcal{T} , we replace h with δ copies (“clones”) of the point h , with one clone associated with each of the δ trajectories that h hits. In total there are at most $k\Delta$ clones. Consider any trajectory $\tau \in \mathcal{T}$, and consider the clones (copies of hit points) that lie along τ . If there are at least 2 clones on τ , then the portion of τ that lies between the extreme clones on τ is captured; the length of this portion is at most the length of τ . The $k\Delta$ clones capture portions of at most $\lfloor k\Delta/2 \rfloor$ trajectories, resulting in total captured length $L^* \leq \ell_1 + \ell_2 + \dots + \ell_{\lfloor k\Delta/2 \rfloor}$, where ℓ_i denotes the length of the i th longest trajectory of \mathcal{T} ($\ell_1 \geq \ell_2 \geq \ell_3 \geq \dots$).

Now consider the simple greedy algorithm that places hit points at the 2 endpoints of the $\lfloor k/2 \rfloor$ longest trajectories of \mathcal{T} , using at most k total hit points. This algorithm captures length $L = \ell_1 + \ell_2 + \dots + \ell_{\lfloor k/2 \rfloor}$. The approximation ratio is at most

$$\frac{L^*}{L} \leq \frac{\lfloor k\Delta/2 \rfloor}{\lfloor k/2 \rfloor}.$$

Thus, $\frac{L^*}{L} \leq \Delta$ for even k . For odd k , the denominator is exactly $(k-1)/2$, while the numerator is either $k\Delta/2$ (if Δ is even) or $(k\Delta-1)/2$ (if Δ is odd); thus, $\frac{L^*}{L} \leq \frac{k\Delta/2}{(k-1)/2} = \Delta(1 + \frac{1}{k-1})$. \square

7.4. Algorithm Engineering

As the TCP can be considered an optimization problem on a weighted graph, we can use approaches such as Integer Programming and local search heuristics. Given the geometric origins of the TCP, we consider geometric aspects; in addition, dealing with geometric data involved a number of other aspects of algorithm engineering, such as accuracy and correctness when handling locations, coordinates, and intersections.

7.4.1. Integer Programming

An IP Formulation

As a problem of combinatorial optimization, the TCP can be modeled as an Integer Program (IP), for which solutions can be computed with the help of powerful IP solvers. The following IP models the TCP.

$$\begin{aligned} \max \quad & \sum_{\tau \in \mathcal{T}, e \in E(\tau)} f(e)x_{\tau,e} \\ & \sum_{v \in V} y_v \leq k && \text{(Constraint 1)} \\ \forall \tau = (v_0, \dots, v_l) \in \mathcal{T} : & \\ \quad \forall i \in 0, \dots, l-1 : & \begin{cases} x_{\tau, v_i v_{i+1}} \leq y_i & \text{if } i = 0, \\ x_{\tau, v_i v_{i+1}} \leq y_i + x_{\tau, v_{i-1} v_i} & \text{else} \end{cases} && \text{(Constraint 2)} \\ \quad \forall i \in 1, \dots, l : & \begin{cases} x_{\tau, v_{i-1} v_i} \leq y_i & \text{if } i = l, \\ x_{\tau, v_{i-1} v_i} \leq y_i + x_{\tau, v_i v_{i+1}} & \text{else} \end{cases} && \text{(Constraint 3)} \\ \forall v \in V, \tau \in e \in \tau : & x_{\tau,e}, y_v \in \{0, 1\} \end{aligned}$$

We have two types of Boolean variables: y_v , for $v \in V$, which indicates if node v is one of the k selected portals, and $x_{\tau,e}$, for edge $e \in E$ on

trajectory τ , which indicates if the portion e of trajectory τ is captured by selected portals. For an edge e , there are distinct variables, $x_{\tau,e}, x_{\tau',e}$, for trajectories $\tau \neq \tau'$, because e can be captured in τ but not in τ' .

Our objective function maximizes the weighted sum of captured trajectory edges, where $E(\tau)$ denotes the edges of τ in \mathcal{G} , and $f(e)$ is the weight (i.e. length) of edge e . (Optionally, we could have trajectory-dependent weights on edges.) Constraint 1 limits the number ($\leq k$) of selected portals. Constraints 2 and 3 enforce that, in order for an edge to be captured as part of trajectory τ , there must be a selected portal in each direction; either there is a selected portal at the next node, or the following trajectory edge is also captured. In the latter case, because τ has no cycle (it is a simple path), there must be a selected portal on τ at some point in that direction if any portion of τ is to be captured.

An IP Example

Figure 7.4 shows an example for the IP formulated in Subsection 7.4.1; using “ x_i ” as shorthand for the IP variables $x_{\tau,v_i v_{i+1}}$ that correspond to the edges along trajectory path $\tau = (v_0, v_1, \dots, v_6)$. Constraints 2 are: $x_0 \leq y_0, x_1 \leq y_1 + x_0, x_2 \leq y_2 + x_1, \dots, x_5 \leq y_5 + x_4$. Constraints 3 are: $x_5 \leq y_6, x_4 \leq y_5 + x_5, x_3 \leq y_4 + x_4, \dots, x_0 \leq y_1 + x_1$. With portals selected at v_1 and v_4 (i.e., with $y_1 = y_4 = 1, y_i = 0$ otherwise), we get constraints $x_0 \leq 0, x_5 \leq 0$, and $x_4 \leq y_5 + x_5 = 0$, implying that only the subpath (v_1, v_2, v_3, v_4) of τ contributes to the objective function.

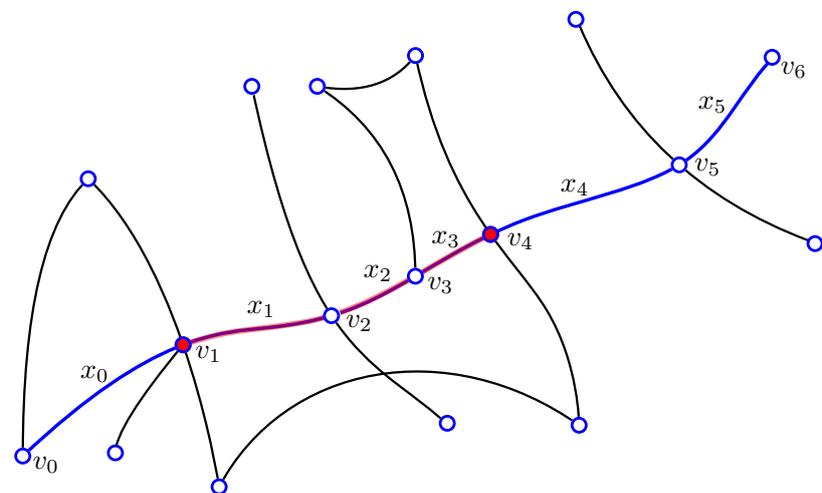


Figure 7.4. Formulating the IP: Trajectory $\tau = (v_0, v_1, \dots, v_6)$ is highlighted in blue. With selected portals at v_1 and v_4 , the portion highlighted in red is captured.

Fractional Solutions

Relaxing the integrality constraints of the IP may result in fractional solutions. We show that the gap (ratio) between the best fractional and the integral (optimal) solution objective functions can be arbitrarily large, for any fixed k .

Theorem 7.4.1 *The integrality gap for the TCP IP can be arbitrarily large for any k .*

Proof. Consider the family of examples that arise from the arrangement of segments corresponding to the embedding of the complete graph, K_n , with its n nodes embedded as evenly spaced points, U , on the boundary of a circle of diameter 1; see Figure 7.5.

The corresponding arrangement graph \mathcal{G} has vertices at the n points U , as well as at the vertices (crossing points) in the arrangement; edges of \mathcal{G} connect consecutive vertices along the segments. The set \mathcal{T} includes a trajectory (of collinear edges) along each of the $\binom{n}{2}$ line segments.

For $k = 1$, the integrality gap is infinite, because the integral solution cannot capture anything, while the fractional solution can capture the edges fractionally. For $k > 1$, the optimal solution can capture less than k^2 trajectories, with a maximal trajectory length of 1, so $OPT_{INT}(k) \leq k^2$. The fractional solution, however, can include fractional portals, with variable values k/n , at each of the n points U , resulting in each trajectory being captured fractionally with value k/n . For simplicity and without loss of generality, assume that n is a multiple of 4. Then, each point U has at least $n/2$ incident segments with length at least 0.5. To see this, simply look onto the left most vertex: Each point of U on the right half has a distance of at least 0.5 (in fact, at least $\sqrt{0.5}$). This means that there are at least $n \cdot \frac{n}{4}$ trajectories with length of at least 0.5, so the overall length of all trajectories is at least $\frac{n^2}{8}$. These trajectories are all captured with fraction at least $1/k$, so $OPT_{FRAC}(k) \geq \frac{1}{k} \cdot \frac{n^2}{8}$. With

$$\frac{OPT_{FRAC}(k)}{OPT_{INT}(k)} \geq \frac{\frac{1}{k} \cdot \frac{n^2}{8}}{k^2} \geq \frac{n^2}{8k^3},$$

we obtain an arbitrarily large integrality gap, as n increases (for any fixed k). □

For instances arising from non-overlapping (i.e., no parallel segments may share more than one point) axis-parallel segments, we can bound the integrality gap, because the particularly bad “clusters” of the general case cannot occur.

Theorem 7.4.2 For trajectories \mathcal{T} arising from non-overlapping axis-parallel line segments, the integrality gap is at most $\frac{k}{\lfloor k/2 \rfloor}$, for $k \geq 2$.

Proof. We can easily get an integral solution by simply capturing the $\lfloor \frac{k}{2} \rfloor$ longest trajectories (segments) by selecting their at most k endpoints as portals.

We create a new LP instance, called LP_2 , by including two copies v_1, v_2 of each portal variable v , so that one copy lies only on horizontal segments, while the other lies only on vertical segments. We constrain $y_{v_1} = y_{v_2} = y_v$ and allow a budget of $2k$ portals for LP_2 . Any feasible values for the y_i in the original LP solution are still feasible in LP_2 (setting both copies), so the optimal solution of LP_2 is an upper bound for the original LP. Because segments do not overlap, every portal now lies only on a single segment. Thus the optimal solution for LP_2 covers the $\frac{2k}{2} = k$ longest segments. This shows the integrality gap is at most $\frac{k}{\lfloor k/2 \rfloor}$. □

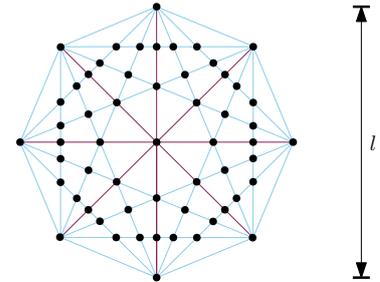


Figure 7.5.: Example used in the proof of Theorem 7.4.1.

The bound of Theorem 7.4.2 is tight for $k = 2$: Consider four segments that are edges of a unit square; then, $k = 2$ portals can capture at most length 1, while a fractional value of $1/2$ at each of the four corners yields objective value $4/2 = 2$ for the LP. For $k \geq 4$, it becomes increasingly difficult to build instances with a high integrality gap.

7.4.2. Heuristics

Integer programming solvers can provide provably optimal or near-optimal solutions for relatively large instances. However, eventually runtime and memory requirements become a limiting factor for large enough instances, so it becomes important to develop effective heuristics. We consider a spectrum of heuristics: *Greedy*, which constructs solutions from scratch by locally optimal choices; *Iterated Local Search*, which iteratively improves a current solution by finding a better one in its local neighborhood; *Simulated Annealing*, which uses a “temperature” function that governs the probability of temporarily accepting a worse solution during a local search; and *Genetic Algorithms*, which maintain a selection of solutions that are locally modified and combined to achieve gradually better solutions.

Greedy begins by selecting the two portals at the ends of the longest trajectory, and then incrementally, greedily selects portals that in each step increase the total captured length as much as possible. *Greedy* can be fooled and give poor solutions; it can, though, serve to give a reasonable starting solution for our other metaheuristics.

Iterated Local Search (ILS) is a basic metaheuristic that, given an initial solution, iteratively replaces the current solution with the best solution found by applying a single local modification, until no further improvement can be achieved. The set of solutions that can be obtained by a single local modification from a specific solution is called its *neighborhood*. For a local modification operator based on changing a single portal, the neighborhood consists of all solutions that differ in exactly one portal. The smaller the neighborhood, the faster the best solution within it can be found; however, a smaller neighborhood also reduces the search space and correspondingly can reduce the quality of the obtained solutions. We consider *global neighborhoods*, based on moving a random portal to an alternative random candidate node, and *local neighborhoods*, based on moving a single portal to positions adjacent to other (unmoved) portals. ILS is initialized with any reasonable solution; after some experimentation with alternatives (e.g., random selection), we settled on using *Greedy* as the starting solution for ILS.

Simulated Annealing (SA) is similar to ILS, but instead of searching for the best solution in the neighborhood, it selects a random solution in the neighborhood and moves to it if (1) it is an improvement, or (2) if it is not an improvement, but it passes a random test. The probability of moving to a worse solution is determined by a “temperature function” and decreases with time. Initially, it can easily escape local optima; when the search satisfies a termination criterion, it returns the best solution it found.

We consider three different termination criteria: the total number of iterations, the number of iterations without an improvement, and the

total runtime. For temperature regulation, we use a geometric reduction by a constant multiplicative amount; for diversification we “reheated” the temperature to the start temperature when we did not change the solution for a certain number of iterations. For translating the temperature function to a probability function, we use the Boltzmann function: $\text{bolzmann}(s', s'', T) := \exp\left(-\frac{s'-s''}{T}\right)$, where s' and s'' are the captured weight of two neighboring solutions. In addition, we use parallelization for pursuing multiple searches from different starting points.

Evolutionary algorithms (EAs) are motivated by the way adaption to environmental conditions happens in nature. They maintain a “population” of current solutions. At each step, the EA produces new solutions through mutations (i.e., local changes) and recombination (combining pieces of solutions in the current population to create new ones). Then, the EA keeps the best solutions (previous or new) to maintain a stable population size. We create the initial population by a version of Greedy that starts with a random segment instead of the longest one. For mutations, we use ILS or SA. The probability of selection for recombination is $\frac{f(s)-f(s_{\min})}{\sum_{s' \in S} f(s')}$. We use uniform random crossover.

7.4.3. Generating Benchmark Instances

Our IP and heuristic methods apply to general sets of trajectories \mathcal{T} , given by spatiotemporal or combinatorial data. We focus on geometric instances, most of which are based on line-segment trajectories. Instances based on random segments tend to be very easy to solve because most vertices have degree 2. So we generated instances based on a set of seed points and selected segments linking them, resulting in arrangement graphs with multi-trajectory intersections and more complicated covering graphs. Alternatively, we tested adding new intersection points to the set of seed points when incrementally constructing the arrangement. For all methods, we used exact intersection point computations from the COMPUTATIONAL GEOMETRY ALGORITHMS LIBRARY (CGAL) to overcome problems of floating point precision for large instances. We generated seed points randomly, using a variety of spatial distributions, including uniform distributions, point sets from the TSP benchmark library [198], and point sets with density distributions based on light maps, corresponding to population densities (see [21]). The distribution of the regular instances can be seen in Figure 7.6.

7.4.4. Experimental Evaluation

All experiments were performed on a single Intel(R) Core(TM) i7-4770 (4×3.4 GHz) with 32 GB and CPLEX (V12.7.1 with default settings), with a time limit of 900 s. The code is available at https://github.com/ahillbs/trajectory_capturing.

Integer and Linear Programming

We first consider the sizes of instances that our IP can solve to optimality within a 900 s time limit, and we consider which factors contribute to the difficulty of the instance.

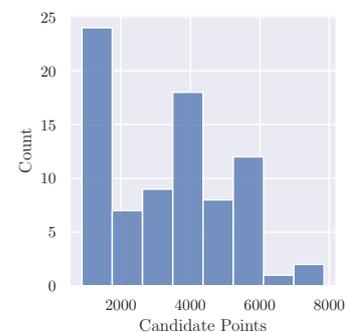
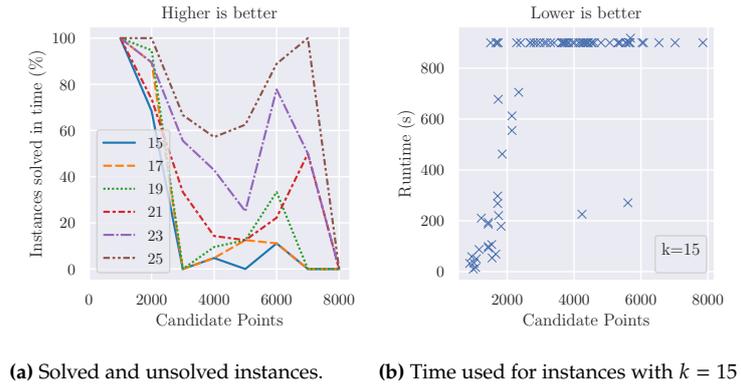


Figure 7.6.: Distribution of regular instances in the benchmark set.

Figure 7.7.: Test results for solved and unsolved instances using the IP. The number of seed points varies from 35 up to 55 points. All tests were performed with a time limit of 900 s.



We varied the number of seed points between 35 and 55 and varied the (uniform) probability of a segment connecting two seeds between 10% and 20%. In Figure 7.7a, we see that instances with up to about 2500 candidate points (i.e., nodes at intersection points in the arrangement, where portals can be placed) can be solved for $15 \leq k \leq 23$ to provable optimality within the time limit. Instances with more than 2500 candidate points are most often not solved for $k < 23$. For instances with $k \geq 23$, the problem seems to be easier to solve. A reason for this may be that these many portals are sufficient to cover all important intersections for many of the considered instances. In Figure 7.7b we see that for $k = 15$ and between 1500 and 2000 candidate points, instances start to become very difficult to solve. However, for $k \geq 23$, instances are still solvable for more than 2500 candidate points. The increase of solved instances in the end can be by chance due to the sparsity of instances with more than 6000 candidate points. However, it may also be that larger instances have more high-yield points with many intersections or clusters which are obvious choices and simplify the portal placement.

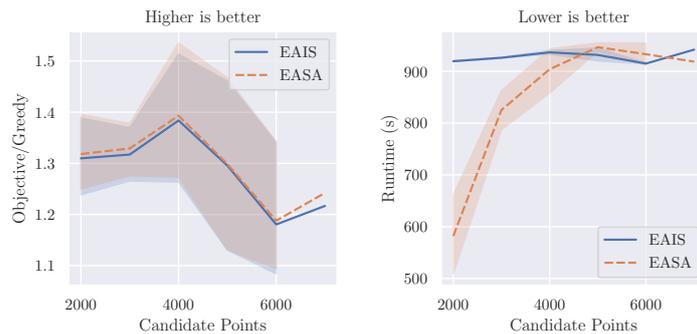
Heuristic Methods

Neighborhoods for Local Methods. For modifying a given solution, we considered global neighborhoods, in which a portal is moved to an arbitrary other position, and local neighborhoods, in which a portal is only moved to positions that connect to another portal. Using global neighborhoods, all solutions are theoretically quickly reachable but they are significantly larger than local neighborhoods and, thus, a meta-heuristic may not work in a focused enough way. Details of this comparison can be found in Section 7.4.4 (Local vs. Global Neighborhoods) on the next page; in particular Figure 7.9 shows our experimental evaluation. Iterated Local Search yields the same solution quality for both neighborhoods; with global neighborhood, only the runtime increases. Simulated Annealing with global neighborhoods barely improves the initial greedy solution, while it gives the best solutions with local neighborhoods.

As a result, we used local neighborhoods for all meta-heuristics.

Mutation Strategy for Evolutionary Algorithms. For evolutionary algorithms, choosing the right kinds of mutations is of crucial importance, as these allow reaching solutions that are not achievable only via recombination. Practical usefulness requires focused mutations that have a

high probability of being useful, instead of purely random changes. That is why we considered Iterated Local Search and Simulated Annealing as mutation operations. As Simulated Annealing has a longer runtime, we used a faster terminating version (with potentially worse solutions) when using it for mutation. In the following we refer to the version with Simulated Annealing as EASA and with Iterated Local Search as EAIS. We have a start with 100 solutions and keep an ongoing population of 50 solutions. The evolutionary algorithm stops after 15 minutes (but can take slightly longer to finish the last round) or if it has not found an improvement for multiple rounds.



(a) Solution quality for Evolutionary Algorithms (b) Runtimes for Evolutionary Algorithms

Figure 7.8.: Comparison of solution quality and runtime by Evolutionary Algorithm with Iterated Local Search and with Simulated Annealing.

Figure 7.8 shows the experimental comparison of both mutation variants. One can see that EASA performs slightly better and is significantly faster for smaller instances. This implies that EASA often quickly finds a good solution but is usually not able to improve it further and terminates early. EAIS, on the other hand, is able to improve its quality until the time limit but still remains slightly worse. For the further experiments, we settled on EASA.

Local vs. Global Neighborhoods

A test for the comparison between Integer Linear Programming and heuristic approaches in Subsection 7.4.4 focuses on choosing good neighborhoods for the meta-heuristics. To this end, we compared the results of local vs. global neighborhoods with respect to quality and time.

Results are shown in Figure 7.9. When using a global neighborhood, it turns out that in most cases, Simulated Annealing does not find better solutions than those provided by greedy. This is most likely due to the fact that the neighborhood space is too large: Looking for a random neighbor will result in a swapped portal that is not connected to another portal via segments, yielding a worse solution. Therefore, the objective value is lowered at high temperatures, while better neighboring solutions will be hard to find at lower temperatures. This is illustrated in Figure 7.9a, where runtimes are lower than for any other heuristic, with the exception of greedy. This is because the algorithm terminates if no improvements of the best solution are found for a certain number of iterations. As a result, global neighborhoods are not recommended for Simulated Annealing.

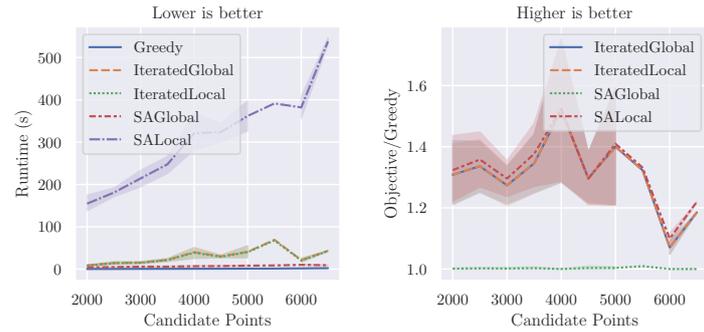


Figure 7.9.: Comparison of solutions for local and global neighborhood with Iterated Local Search and Simulated Annealing for $k = 25$.

(a) Comparison of required time

(b) Comparison of achieved objective values and greedy solutions

On the other hand, the results with local neighborhoods are better with respect to the objective value. However, Simulated Annealing with local neighborhood also requires the largest runtimes by a wide margin, in particular for large instances, as shown in Figure 7.9a. This can be explained by the “reheating” process after low temperatures: The algorithm tries to find better solutions than the current solution for a while; if no better solution is found for a fixed number of iterations, it reheats the temperature to escape a local optimum, allowing worse solutions as the current one. As the process continues when an improvement is found, the resulting gradual search process may continue for a long time.

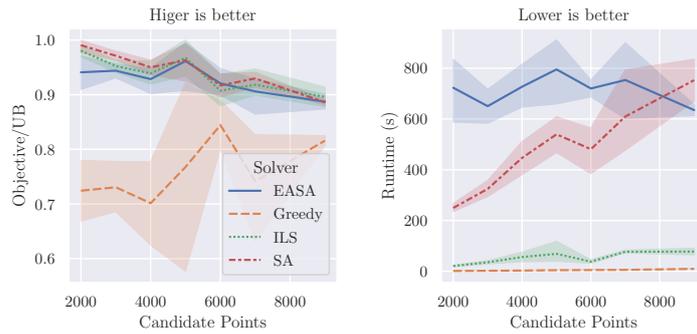
Both the local and the global variants of Iterated Local Search produce almost always the same solution values, as shown in Figure 7.9b. This is due to the fact that both are looking for the best neighbor in every iteration. Because a swapped portal needs to be connected to one or more portals via segments, the local neighborhood consists of all vertices that are connected to at least one portal via segments. Therefore, both search variants produce the same solutions almost every time. This makes local Iterated Local Search slightly preferable: As shown in Figure 7.9a, it is marginally faster than global Iterated Local Search by a small margin, due to smaller search space. (This effect is not more pronounced because the search for a locally best neighbor seems to be almost as slow as for a globally best neighbor.)

Overall local neighborhoods appear to perform better than global ones, which is why they were selected for the following tests.

Comparison of Heuristics with IP as Baseline

We compared the heuristics in terms of solution quality and runtime against the IP solver, which produces not only solutions, but also guaranteed bounds.

Figure 7.10 shows the obtained results. The Evolutionary Algorithm produces, on average, the worst solutions of all metaheuristics, while still requiring more time than the others. However, it is the only metaheuristic tested that reliably computes good solutions for instances consisting of several point clusters, for which solutions consist of several connected components. These instances did not occur with our generation method



(a) Solution quality for heuristics and IP solutions. (b) Runtimes of heuristics and IP solutions.

Figure 7.10.: Comparisons of solution quality and runtime for all tested algorithms.

but only in separate, manually created instances which are not part of this experiment.

The greedy approach never falls below $\frac{1}{2}$ OPT for these instances, while being the fastest.

Iterated Local Search (ILS) appears to produce quite good solutions, which are not worse than 10% below the optimal solution, in a very short time frame. While it only finds a local optimum, it seems that the objective function quality of these are quite close to the optimal values. As a consequence, Iterated Local Search can produce good solutions for instances with up to 5000 candidate points and $k = 25$.

The best heuristic algorithm, in terms of solution quality and runtime, appears to be Simulated Annealing. The combination of fast diversification at high temperature and random swaps for improving solutions at low temperature seems to work quite well; in addition, the mechanism of reheating to restart diversification, in combination with multi-threading to cover a larger search space, are characteristics that are not present in the Evolutionary Algorithm. For $k = 25$, Simulated Annealing produces excellent solutions for instances with up to 5000 candidate points; this instance size can be easily increased for smaller k .

In summary, we can produce excellent heuristic solutions for instances with up to 5000 candidate points and $k = 25$. If fast solutions are desired, Iterated Local Search is the method of choice. The best tradeoff between runtime and solution quality is offered by Simulated Annealing. Finally, for cluster-based instances, we recommend Evolutionary Algorithms.

Linear Programming and Integrality Gap

As described in Section 7.3, if the TCP input trajectories come from K subsets of noncrossing trajectories, we have a K -approximation algorithm based on dynamic programming. In particular, if the input trajectories consist of axis-parallel line segments, $K = 2$, so there is a 2-approximation. This may coincide with better practical solvability of these kinds of instances. We have verified this for some instances for which all segments are axis-parallel and (for collinear segments) non-overlapping. (See Figure 7.11 for such an instance with 1100 segments and roughly 8200 to 8500 candidate points. We have also solved instances with 2000 segments and 19 000 candidate points.) Furthermore, for instances with up to 20 000

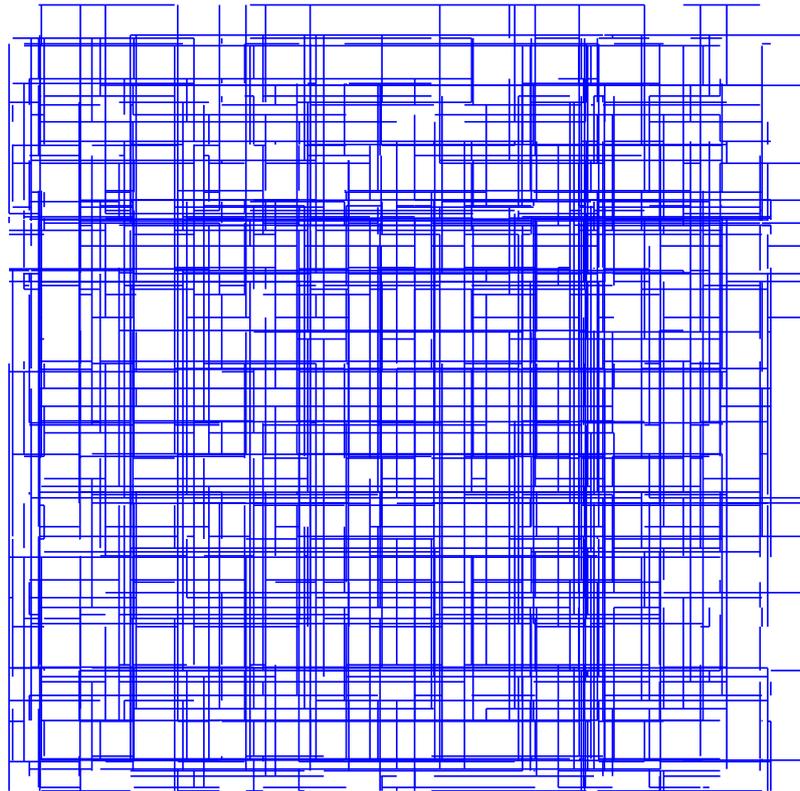


Figure 7.11.: An instance with 1100 non-degenerate axis-parallel segments

points, the integrality gap was never larger than 20% for $k = 5$, 7% for $k = 10$, 5% for $k = 15$ and less than 2.5% for larger k .

Application to Taxi Trajectory Data

We have applied our TCP model to solve real-world data sets to optimality. In Figure 7.12 we show the results of computing $k = 5$ optimal portals for a set of trajectories based on taxi cab routes in the San Francisco Bay Area. The data is based on 375 vehicles, sampled every 5 min, 288 times per day, for one week [199]; see the trajectories in Figure 7.1.

Our experiments included runs on 30 instances, with k ranging from 5 to 11, on sets of 10 to 120 trajectories of varying lengths (comprised of 1300 to 3700 edges, and 600 to 1800 vertices). The trajectories are snapped to a regular grid graph. Solution times of the IP were up to 200 seconds of computation, with most instances taking less than 10 seconds.

7.5. Conclusion

We have introduced the trajectory capture problem (TCP), an optimization problem in which we seek to place k points (or portals) in order to “capture” the maximum total length of a given set of paths/trajectories between two placed points. We have shown that the problem is NP-hard, even for axis-aligned line segment trajectories in the plane, and we have given approximation algorithms for two cases. Can we improve the approximation factor of K for a set of trajectories that is the union of K

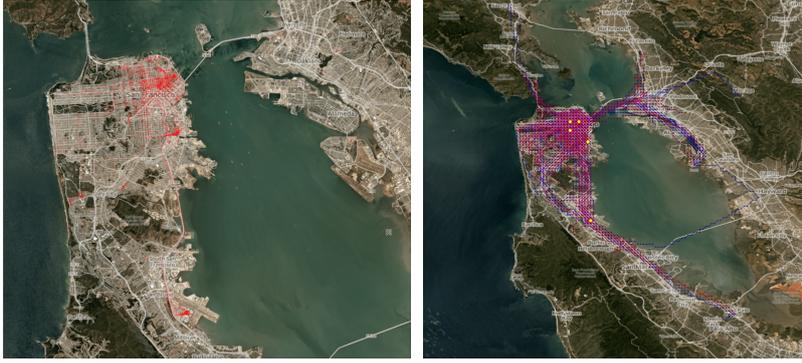


Figure 7.12.: (Left) Candidate points before processing. (Right) Solution to real-world TCP instance: An optimal set of $k = 5$ portals are highlighted. Red trajectory portions (some of which may loop back) are captured, blue ones are not captured. Satellite images are courtesy of Planet Labs Inc.

subsets, each of which is noncrossing? Can we improve the approximation factor of Δ for a set of trajectories of depth at most Δ ?

A focus of our work is the exploration, via algorithm engineering, of practical methods for solving the TCP. Our methods are based on integer programming and on simple heuristic search methods. It will be interesting to develop more specified methods for other, specific classes of instances, such as further geometric instances arising from other types of real-world geographic data.

Part IV.

TILT PROBLEMS

In the first part of this thesis, we considered satellites in space. In the second part, we considered drones and robots on earth. In this fourth part, we consider controlling tiny particles with applications within the human body.

At a specific size, agents can no longer actuate themselves, rather need to be controlled by external forces. While the volume and energy capacity of an agent decrease cubically with its size, the surface and fraction only decrease quadratically. If the agents cannot carry enough energy to move themselves, we can move them by applying external, e.g., magnetic forces. This gives rise to new geometric problems, which we call *tilt problems*.

In Chapter 8 (Tilt Assembly) on page 177, we consider how to construct miniature shapes by iteratively adding sticky particles from the outside using such an external force. It is not possible for all shapes to be constructed this way, and finding a construction sequence is a difficult problem. We provide a set of theoretical results and use a SAT-solver to find construction sequences for shapes consisting of more than 500 particles.

In Chapter 9 (Targeted Drug Delivery) on page 203, we try to use the external forces to guide a set of particles to a specific location, e.g., to a tumour for targeted drug delivery. The challenge is that the external force actuates all particles equally and manipulations are only possible by collisions with the environment. We show that the involved problem is NP-hard even in two-dimensional environments, but we also provide algorithms with performance guarantees. However, in our computational study, reinforcement learning performs significantly better than the classical algorithmic approaches and local heuristics. Thus, we also take a look at how we can solve optimization problems with reinforcement learning algorithms.

This chapter presents algorithmic results for the parallel assembly in two and three dimensions of many micro-scale objects from tiny particles, which has been proposed in the context of programmable matter and self-assembly for building high-yield micro-factories. The underlying model has particles moving under the influence of uniform external forces until they hit an obstacle. Particles bond when forced together with another appropriate particle. Aside from a number of theoretical results and tools, we discuss how to use a SAT-solver to compute construction sequences for shapes with more than 500 particles or to prove infeasibility.

8.1. Introduction

In recent years, progress on flexible construction at micro- and nano-scale has given rise to a large set of challenges that deal with algorithmic aspects of programmable matter. Examples of cutting-edge application areas with a strong algorithmic flavor include self-assembling systems, in which chemical and biological substances such as DNA are designed to form predetermined shapes or carry out massively parallel computations; and swarm robotics, in which complex tasks are achieved through the local interactions of robots with severely limited individual capabilities, including micro- and nano-robots.

Moving individual particles to their appropriate attachment locations when assembling a shape is difficult because the small size of the particles limits the amount of onboard energy and computation. One successful approach to dealing with this challenge is to use molecular diffusion in combination with cleverly designed sets of possible connections: in *DNA tile self-assembly*, the particles are equipped with sophisticated bonds that ensure only a predesigned shape is produced when mixing together a set of tiles, see [200]. The resulting study of algorithmic tile self-assembly has given rise to an extremely powerful framework and produced a wide range of impressive results. However, the required properties of the building material (which must be specifically designed and finely tuned for each particular shape) in combination with the construction process (which is left to chemical reactions, so it cannot be controlled or stopped until it has run its course) make DNA self-assembly unsuitable for some applications.

An alternative method for controlling the eventual position of particles is to apply a uniform external force, causing all particles to move in a given direction until they hit an obstacle or another blocked particle. Becker et al. [201] have shown, combining this approach with custom-made obstacles (instead of custom-made particles) allows complex rearrangements of particles, even in grid-like environments with axis-parallel motion. The appeal of this approach is that it shifts the design complexity from

8.1 Introduction	177
8.1.1 Overview	179
8.1.2 Related Work	179
8.2 Preliminaries	181
8.3 Constructibility of Simple Polyominoes	181
8.3.1 A Key Lemma	182
8.3.2 An Efficient Algorithm	184
8.3.3 Pipelined Assembly	185
8.3.4 Error Detection	187
8.4 Optimization Variants	190
8.5 Three-dimensional Shapes	193
8.6 Computational Study	196
8.6.1 Solving as SAT-Formula	196
8.6.2 Constraint Programming	198
8.6.3 Evaluation	198
8.7 Conclusion	201

*This chapter is based on [8]. Subsection 8.3.4 is a part of [9]. Section 8.6 is new content.

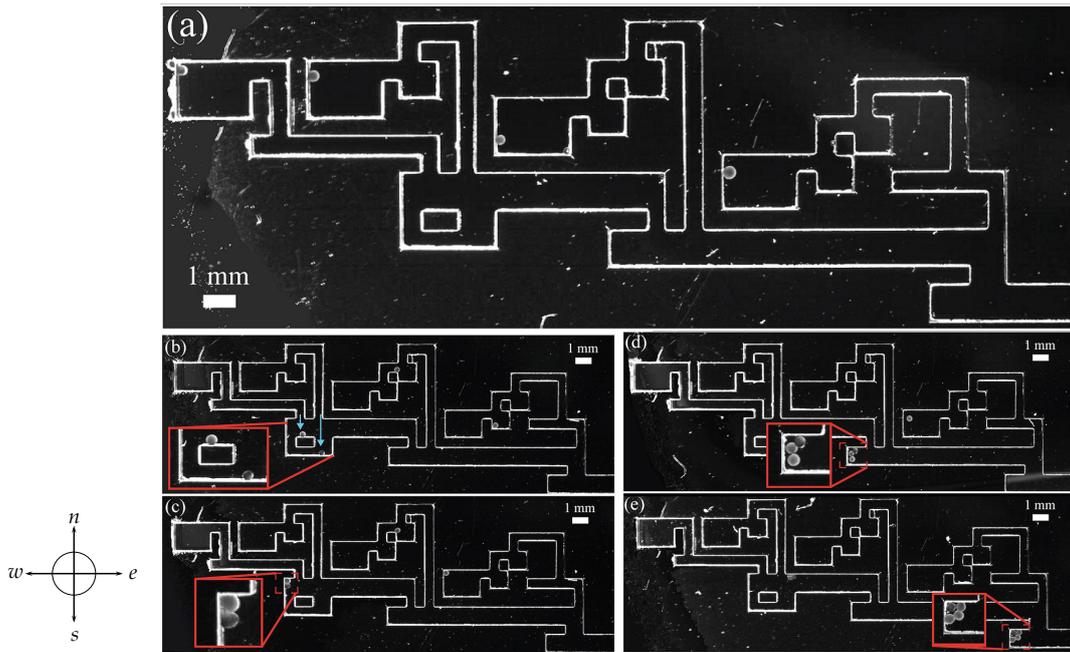


Figure 8.1: A practical demonstration of Tilt Assembly based on alginate (i.e., a gel made by combining a powder derived from seaweed with water) particles on a silicon wafer with obstacles etched in photoresist [202]. The obstacles appear as white lines and block the motion of particles. (a) Alginate particles in initial positions. (b) After control moves of $\langle e, s, w, n, e, s \rangle$ (for east, south, west, north), the alginate microrobots move to the shown positions. (c) After $\langle w, n \rangle$ inputs, the system produces the first multi-microrobot polyomino. (d) The next three microrobot polyominoes are produced after applying multiple $\langle e, s, w, n \rangle$ cycles. (e) After the alginate microrobots have moved through the microfluidic factory layout, the final 4-particle polyomino is generated. This figure originates from [202].

the building material (the tiles) to the machinery (the environment). As practical work by Manzoor et al. [202] shows, it is possible to apply this to simple “sticky” particles that can be forced to bond, see Figure 8.1: the overall assembly is achieved by adding particles one at a time, attaching them to the existing sub-assembly.

Moreover, Manzoor et al. [202] argue that it is possible to enhance the overall assembly environment to allow pipelined construction, as shown in Figure 8.3: after constructing the first polyomino, each cycle of a small control sequence produces another polyomino. However, the algorithmic part of [202] is purely heuristic; providing a thorough understanding of algorithms and complexity is the content of this chapter.

One critical issue of this approach is the requirement of getting particles to their destination without being blocked by or bonding to other particles. As Figure 8.2 shows, this is not always possible, so there are some shapes that cannot be constructed by Tilt Assembly.

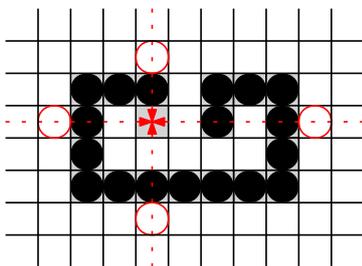


Figure 8.2: A polyomino (black) that cannot be constructed by Tilt Assembly: the last tile cannot be attached, as it gets blocked by previously attached tiles.

This gives rise to a variety of algorithmic questions:

1. Can we decide efficiently whether a given polyomino can be constructed by Tilt Assembly?
2. Can the resulting process be pipelined to yield low amortized building time?
3. Can we compute a maximum-size subpolyomino that can be constructed?
4. What can be said about three-dimensional versions of the problem?

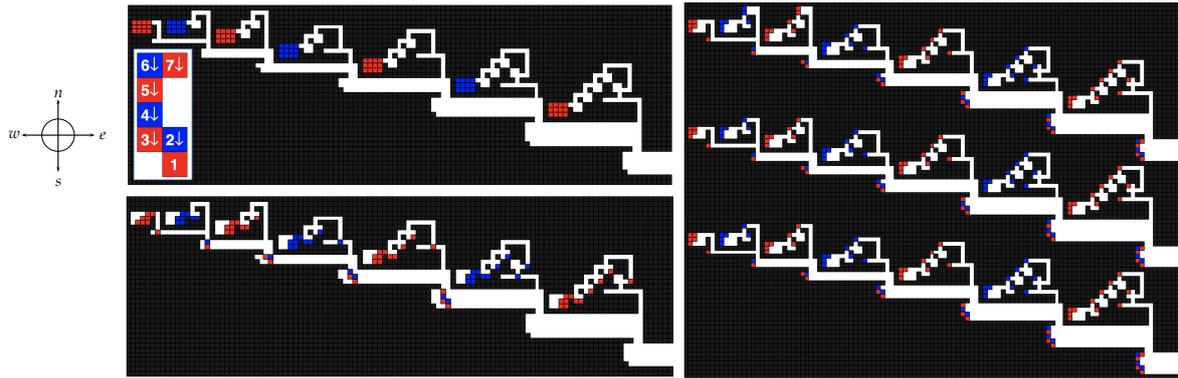


Figure 8.3.: (Top left) Initial setup of a seven-tile polyomino assembly; the composed shape is shown enlarged on the lower left. The bipartite decomposition into blue and red particles is shown for greater clarity, but can also be used for better control of bonds. The sequence of control moves is $\langle e, s, w, n \rangle$, i.e., a clockwise order. (Bottom left) The situation after 18 control moves. (Right) The situation after 7 full cycles, i.e., after 28 control moves; shown are three parallel “factories”. This figure originates from [202].

8.1.1. Overview

The main result of this chapter is a characterization of deciding constructibility and efficient construction for simply connected two-dimensional shapes in Section 8.3: For a simple polyomino P with N pixels, we can decide in time $O(N \log N)$ whether P can be constructed; using pipelining, a constructible polyomino can be built in amortized time $O(1)$. On the other hand, we show that deciding constructibility in 3D is NP-complete in Section 8.5. We also provide a number of additional results on approximation and the constructibility of subpaths; see Table 8.1 for an overview. Finally, we perform a computational study in Section 8.6 showing how we can prove or disprove constructibility of arbitrary two-dimensional and three-dimensional instances with more than 500 tiles using a SAT-solver.

Dimension	2D	3D
Polyomino	simple	general
Decision	$O(N \log N)$ (Sec. 8.3)	NP-complete (Sec. 8.5)
Maximization	<i>polyAPX-hard</i>	<i>polyAPX-hard</i>
Approximation	$O(N^{1/3}), \Omega(\sqrt{N})$ (Sec. 8.4)	$O(N^{1/3}), -$ (Sec. 8.4)
Constructible Path	$O(N \log N)$ (Sec. 8.4)	NP-complete (Sec. 8.5)

Table 8.1.: Results for Tilt Assembly Problem (TAP) and its maximization variant (MAXTAP)

8.1.2. Related Work

Assembling polyominoes with tiles has been considered intensively in the context of *tile self-assembly*. In 1998, Erik Winfree [200] introduced the *abstract tile self-assembly model* (aTAM), in which tiles have glue types on each of the four sides and two tiles can stick together if their glue type matches and the bonding strength is sufficient. Starting with a *seed tile*, tiles will continue to attach to the existing partial assembly until they form a desired polyomino; the process stops when no further attachments are possible. For early examples of related work, see Rothmund and Winfree [203] and Adleman et al. [204] for the running time and program size for self-assembling squares. Apart from the aTAM, there are various other models like the *two-handed tile self-assembly model* (2HAM) [205] and the *hierarchical tile self-assembly model* [206], in which we have no single seed but pairs of subassemblies that can attach to each other. Furthermore, the

staged self-assembly model [207–209] allows greater efficiency by assembling polyominoes in multiple bins which are gradually combined with the content of other bins.

All this differs from the model in Tilt Assembly, in which each tile has the same glue type on all four sides, and tiles are added to the assembly one at a time by attaching them from the outside along a straight line. This approach of externally movable tiles has been considered in practice at the microscale level using biological cells and an MRI, see [210–212]. Becker et al. [213] consider this for the assembly of a magnetic *Gauß gun*, which can be used for applying strong local forces by very weak triggers, allowing applications such as micro-surgery.

Using an external force for moving the robots becomes inevitable at some scale because the energy capacity decreases faster than the energy demand. A consequence is that all non-fixed robots/particles perform the same movement, so all particles move in the same direction of the external force until they hit an obstacle or another particle. These obstacles allow shaping the particle swarm. Designing appropriate sets of obstacles and moves gives rise to a range of algorithmic problems. Deciding whether a given initial configuration of particles in a given environment can be transformed into a desired target configuration is NP-hard [201], even in a grid-like setting, whereas finding an optimal control sequence is shown to be PSPACE-complete by Becker et al. [214]. However, if *designing* the obstacles is allowed, the problems become more tractable [201]. Moreover, even complex computations become possible: If we allow additional particles of double size (i.e., two adjacent fields), full computational complexity is achieved, see Shad et al. [215]. Further related work includes gathering a particle swarm at a single position [22] and using swarms of very simple robots (such as Kilobots) for moving objects [216]. For the case in which human controllers have to move objects by such a swarm, Becker et al. [217] study different control options. The results are used by Shahrokhi and Becker [218] to investigate an automatic controller.

The construction of polyominoes has also applications in the field of robot swarms, e.g., *shape formation*. Werfel and Nagpal [219, 220] show how multiple robots can move tiles to a partial assembly to construct a desired shape in 2D and 3D. Derakhshandeh et al. [221, 222] consider the robots as building material, which have $O(1)$ memory, and provide algorithms letting the robots form or coat shapes. In a recent paper, Demaine et al. [223, 224] show that a robot swarm can be reconfigured in time $O(d)$ unit steps, where d is the maximum distance of any robot. However, this requires the robots to be well separable. Arbuckle and Requicha [225] show how a self-organized swarm of robots can construct a certain shape. In case of robot failures or external disturbance, the swarm is also able to repair the shape.

Further related work includes robots performing various tasks: Thubagere et al. [226] show that robots made from DNA can simultaneously sort molecular cargoes. Rubenstein et al. [227] consider a swarm of simple robots moving an object to a desired destination without knowing its shape and weight. Hoffmann [228] proves that it is NP-hard to decide if a robot can push its way through an area filled with blocks.

8.2. Preliminaries

In this section we define our model used throughout this paper.

Polyomino: For a set $P \subset \mathbb{Z}^2$ of N grid points in the plane, the graph G_P is the induced grid graph, in which two vertices $p_1, p_2 \in P$ are connected if they are at unit distance. Any set P with connected grid graph G_P gives rise to a *polyomino* by replacing each point $p \in P$ by a unit square centered at p , which is called a *tile*; for simplicity, we also use P to denote the polyomino when the context is clear, and refer to G_P as the dual graph of the polyomino; P is *tree-shaped* if G_P is a tree. A polyomino is called *hole-free* or *simple* if and only if the grid graph induced by $\mathbb{Z}^2 \setminus P$ is connected.

Blocking sets: For each point $p \in \mathbb{Z}^2$ we define *blocking sets* $N_p, S_p \subseteq P$ as the set of all points $q \in P$ that are above or below p and $|p_x - q_x| \leq 1$. Analogously, we define the blocking sets $E_p, W_p \subseteq P$ as the set of all points $q \in P$ that are to the right or to the left of p and $|p_y - q_y| \leq 1$.

Construction step: A *construction step* is defined by a direction $d \in \{\text{north, east, south, west}\}$ (abbreviated by n, e, s, w) from which a tile is added and a latitude/longitude l describing a column or row. The tile arrives from (l, ∞) for north, (∞, l) for east, $(l, -\infty)$ for south, and $(-\infty, l)$ for west into the corresponding direction until it reaches the first grid position that is adjacent to one occupied by an existing tile. If there is no such tile, the polyomino does not change. We note that a position p can be added to a polyomino P if and only if there is a point $q \in P$ with $\|p - q\|_1 = 1$ and one of the four blocking sets, N_p, E_p, S_p or W_p , is empty. Otherwise, if none of these sets are empty, this position is *blocked*.

Constructibility: Beginning with a seed tile at some position p , a polyomino P is *constructible* if and only if there is a sequence $\sigma = ((d_1, l_1), (d_2, l_2), \dots, (d_{N-1}, l_{N-1}))$, such that the resulting polyomino P' , induced by successively adding tiles with σ , is equal to P . We allow the constructed polyomino P' to be a translated copy of P . Reversing σ yields a *decomposition sequence*, i.e., a sequence of tiles removed from P .

8.3. Constructibility of Simple Polyominoes

In this section we focus on hole-free (i.e., simple) polyominoes. We show that the problem of deciding whether a given polyomino can be constructed can be solved in polynomial time. This decision problem can be defined as follows.

Definition 8.3.1 (TILT ASSEMBLY PROBLEM) *Given a polyomino P , the TILT ASSEMBLY PROBLEM (TAP) asks for a sequence of tiles constructing P , if P is constructible.*

8.3.1. A Key Lemma

A simple observation is that construction and (connectivity-preserving) decomposition are the same problem. This allows us to give a more intuitive argument, as it is easier to argue that we do not lose connectivity when removing tiles than it is to prove that we do not block future tiles.

Theorem 8.3.1 *A polyomino P can be constructed if and only if it can be decomposed using a sequence of tile removal steps that preserve connectivity. A construction sequence is a reversed decomposition sequence.*

Proof. To prove this theorem, it suffices to consider a single step. Let P be a polyomino and t be a tile that is removed from P into some direction l , leaving a polyomino P' . Conversely, adding t to P' from direction l yields P , as there cannot be any tile that blocks t from reaching the correct position, or we would not be able to remove t from P in direction l . \square

For hole-free polyominoes we can efficiently find a construction/decomposition sequence if one exists. The key insight is that one can greedily remove *locally convex* tiles. A tile t is said to be locally convex if and only if it is locally extremal, i.e., there are two axis-parallel orthogonal directions for which there is no tile connected to t ; see Figure 8.4. If a locally convex tile is *not* a cut tile, i.e., it is a tile whose removal does *not* disconnect the polyomino, its removal does not interfere with the decomposability of the remaining polyomino.

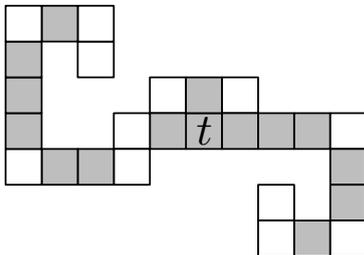


Figure 8.4.: A polyomino and its locally convex tiles (white). Removing the non locally convex tile t destroys decomposability. The polyomino can be decomposed by starting with the three tiles above t .

This conclusion is based on the observation that a minimal cut (i.e., a minimal set of vertices whose removal leaves a disconnected polyomino) of cardinality two in a hole-free polyomino always consists of two (possibly diagonally) adjacent tiles. Furthermore, we can always find such a removable locally convex tile in any decomposable hole-free polyomino. This allows us to devise a simple greedy algorithm.

We start by showing that if we find a non-blocked locally convex tile that is not a cut tile, we can simply remove it. It is important to focus on locally convex tiles, as the removal of not locally convex tiles can harm the decomposability: see Figure 8.4 for an illustration. In non-simple polyominoes, the removal of locally convex tiles can destroy decomposability, as demonstrated in Figure 8.5.

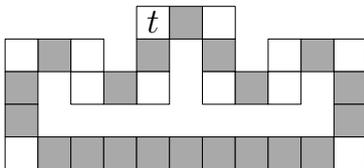


Figure 8.5.: With non-simple polygons we may not be able to remove locally convex tiles. Removing the locally convex tile t leaves the polyomino non-decomposable; it can be decomposed by starting from the bottom or the sides.

Lemma 8.3.2 *Consider a non-blocked, non-cut, locally convex tile t in a hole-free polyomino P . The polyomino $P - t$ is decomposable if and only if P is decomposable.*

Proof. The first direction is trivial: if $P - t$ is decomposable, P is decomposable as well, because we can remove the non-blocked tile t first and afterwards use the existing decomposition sequence for $P - t$. The other direction requires some case distinctions. Suppose for contradiction that P is decomposable but $P - t$ is not, i.e., t is important for the later decomposition.

Consider a valid decomposition sequence for P and the first tile t' we cannot remove if we were to remove t in the beginning. W.l.o.g., let t' be the first tile in this sequence (removing all previous tiles obviously does not destroy the decomposability). When we remove t first, we are missing a tile, hence t' cannot be blocked but has to be a cut tile in the remaining polyomino $P - t$. The presence of t preserves connectivity, i.e., $\{t, t'\}$ is a minimal cut on P . Because P has no holes, then t and t' must be diagonal neighbors, sharing the neighbors a and b . Furthermore, by definition neither of t and t' is blocked in some direction. We make a case distinction on the relation of these two directions.

The directions are orthogonal (Figure 8.6a). Either a or b is a non-blocked locally convex tile, because t and t' are both non-blocked; w.l.o.g., let this be a . It is easy to see that independent of removing t or t' first, after removing a we can also remove the other one.

The directions are parallel (Figure 8.6b). This case is slightly more involved. By assumption, we have a decomposition sequence beginning with t' . We show that swapping t' with our locally convex tile t in this sequence preserves feasibility.

The original sequence has to remove either a or b before it removes t , as otherwise the connection between the two is lost when t' is removed first. After either a or b is removed, t becomes a leaf and can no longer be important for connectivity. Thus, we only need to consider the sequence until either a or b is removed. The main observation is that a and b block the same tiles as t or t' , except for tile c as in Figure 8.6b. However, when c is removed, c has to be a leaf, because a is still not removed and in the original decomposition sequence, t' has already been removed. Therefore, a tile $d \neq t'$ would have to be removed before c . Hence, the decomposition sequence remains feasible, concluding the proof. \square

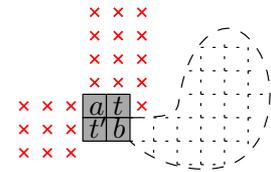
Next we show that such a locally convex tile always exists if the polyomino is decomposable.

Lemma 8.3.3 *Let P be a decomposable polyomino. Then there exists a locally convex tile that is removable without destroying connectivity.*

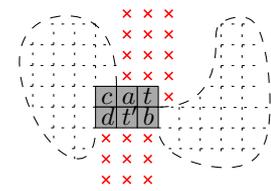
Proof. We prove this by contradiction based on two possible cases.

Assume P to be a decomposable polyomino in which no locally convex tile is removable. Because P is decomposable, there exists some feasible decomposition sequence S . Let P_{convex} denote the set of locally convex tiles of P and let $t \in P_{\text{convex}}$ be the first removed locally convex tile in the decomposition sequence S . By assumption, t cannot be removed yet, so it is either blocked or a cut tile.

t is blocked. Consider the direction in which we would remove t . If it does not cut the polyomino, the last blocking tile has to be locally convex (and would have to be removed before t), see Figure 8.7a. If it cuts the polyomino, the component cut off also must have a locally convex tile and the full component has to be removed before t , see Figure 8.7b. This is again a contradiction to t being the first locally convex tile to be removed in S .

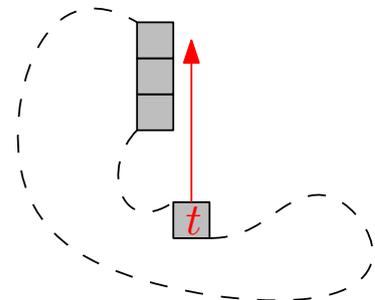


(a) If the unblocked directions of t and t' are orthogonal, one of the two adjacent tiles (w.l.o.g. a) cannot have any further neighbors. There can also be no tiles in the upper left corner, because the polyomino cannot cross the two free directions of t and t' (red marks).

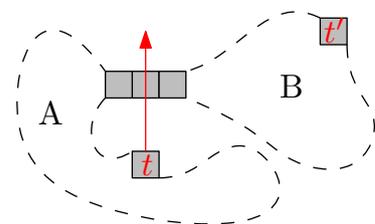


(b) If the unblocked directions of t and t' are parallel, there is only the tile c for which something can change if we remove t before t' .

Figure 8.6: The red marks indicate that no tile is at this position; the dashed outline represents the rest of the polyomino.



(a) If the removal direction of t is not crossed, the last blocking tile has to be locally convex (and has to be removed before t).



(b) If the removal direction of t crosses P , then P gets split into components A and B . Component B has a locally convex tile t' that needs to be removed before t .

Figure 8.7: Polyominoes for which no locally convex tile should be removable, showing the contradiction to t being the first blocked locally convex tile in P removed.

t is a cut tile. $P - t$ consists of exactly two connected polyominoes, P_1 and P_2 . It is easy to see that $P_1 \cap P_{\text{convex}} \neq \emptyset$ and $P_2 \cap P_{\text{convex}} \neq \emptyset$, because every polyomino of size $n \geq 2$ has at least two locally convex tiles of which at most one ceases to be locally convex by adding t . (A polyomino of size 1 is trivial.) Before being able to remove t , either P_1 or P_2 has to be completely removed, including their locally convex tiles. This is a contradiction to t being the first locally convex tile in S to be removed. \square

8.3.2. An Efficient Algorithm

An iterative combination of these two lemmas proves the correctness of greedily removing locally convex tiles. As we show in the next theorem, using a search tree technique allows an efficient implementation of this greedy algorithm.

Theorem 8.3.4 *A hole-free polyomino can be checked for decomposability/constructibility in time $O(N \log N)$.*

Proof. Lemma 8.3.2 allows us to remove any locally convex tile, as long as it is not blocked and does not destroy connectivity. Applying the same lemma on the remaining polyomino iteratively creates a feasible decomposition sequence. Lemma 8.3.3 proves that this is always sufficient. If and only if we can at some point no longer find a matching locally convex tile (to which we refer as *candidates*), the polyomino cannot be decomposable.

Let B be the time needed to check whether a tile t is blocked. A naïve way of doing this is to try out all tiles and check if t gets blocked, requiring time $O(N)$. With a preprocessing step, we can decrease B to $O(\log N)$ by using $O(N)$ binary search trees for searching for blocking tiles and utilizing that removing a tile can change the state of at most $O(1)$ tiles. For every vertical line x and horizontal line y going through P , we create a balanced search tree, i.e., for a total of $O(N)$ search trees. An x -search tree for a vertical line x contains tiles lying on x , sorted by their y -coordinate. Analogously define a y -search tree for a horizontal line y containing tiles lying on y sorted by their x -coordinate. We iterate over all tiles $t = (x, y)$ and insert the tile in the corresponding x - and y -search tree with a total complexity of $O(N \log N)$. Note that the memory complexity remains linear, because every tile is in exactly two search trees. To check if a tile at position (x', y') is blocked from above, we can simply search in the $(x' - 1)$ -, x' - and $(x' + 1)$ -search tree for a tile with $y > y'$. We analogously perform search queries for the other three directions, and thus have 12 queries of total cost $O(\log N)$.

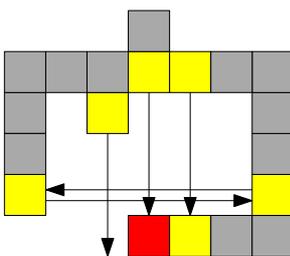


Figure 8.8.: When removing the red tile (dark gray in grayscale), only the yellow tiles (light gray in grayscale) can become unblocked or locally convex.

We now iterate on all tiles and add all locally convex tiles that are not blocked and are not a cut tile to the set F (cost $O(N \log N)$). Note that checking whether a tile is a cut tile can be done in constant time, because it suffices to look into the local neighborhood. While F is not empty, we remove a tile from F , from the polyomino, and from its two search trees in time $O(\log N)$. Next, we check the up to 12 tiles that could have been blocked by the removed tile, see Figure 8.8.

Only these tiles can become unblocked or a locally convex tile. Those that are locally convex tiles, not blocked, and not a cut tile are added to F . All tiles behind those cannot become unblocked as the first tiles would still be blocking them. If one of those tiles becomes a cut tile, then we remove it from F . The cost for this is again in $O(\log N)$. This is continued until F is empty, which takes at most $O(N)$ loops each of cost $O(\log N)$. If the polyomino has been decomposed, the polyomino is decomposable/constructible by the corresponding tile sequence. Otherwise, there cannot exist such a sequence. A specific start tile can be enforced by prohibiting the removal of that tile. \square

8.3.3. Pipelined Assembly

Given that a construction is always possible based on adding locally convex corners to a partial construction, we can argue that the heuristic idea of Manzoor et al. [202] for pipelined assembly can be formally realized for *every* constructible polyomino: we can transform the construction sequence into a spiral-shaped maze environment, as illustrated in Figure 8.9. This allows it to produce D copies of P in $N + D$ cycles, implying that we only need $2N$ cycles for N copies. It suffices to use a clockwise order of four unit steps (west, north, east, south) in each cycle.

The main idea is to create a spiral in which the assemblies move from the inside to the outside. The first tile is provided by an initial south movement. After each cycle, ending with a south movement, the next seed tile of the next copy of P is added. For every direction corresponding to the direction of the next tile added by the sequence, we place a tile depot on the outside of the spiral, with a straight-line path to the location of the corresponding attachment.

Theorem 8.3.5 *Given a construction sequence $\sigma := ((d_1, l_1), (d_2, l_2), \dots, (d_{N-1}, l_{N-1}))$ that constructs a polyomino P , we can construct a maze environment for pipelined tilt assembly, such that constructing D copies of P needs $O(N + D)$ unit steps. In particular, constructing one copy of P can be done in amortized time $O(1)$.*

Proof. Consider the construction sequence σ , the movement sequence ζ consisting of N repetitions of the cycle (w, n, e, s) , and an injective function $m : \sigma \rightarrow \zeta$, with $m((w, \cdot)) = e$, $m((n, \cdot)) = s$, $m((e, \cdot)) = w$ and $m((s, \cdot)) = n$. We also require that $m((d_i, l_i)) = \zeta_j$ if for all $i' < i$ there is a $j' < j$ with $m((d_{i'}, l_{i'})) = \zeta_{j'}$ and j is the smallest possible. This implies that in each cycle there is at least one tile in σ mapped to one direction in this cycle.

Labyrinth construction: The main part of the labyrinth is a spiral as can be seen in Figure 8.9. Consider a spiral that is making $|\zeta|$ many turns, and the innermost point q of this spiral. From q upwards, we make a lane through the spiral until we are outside the spiral. At this point we add a depot of tiles, such that after each south movement a new tile comes out of the depot (this can easily be done with bottleneck constructions as seen in Figure 8.9). Then, we

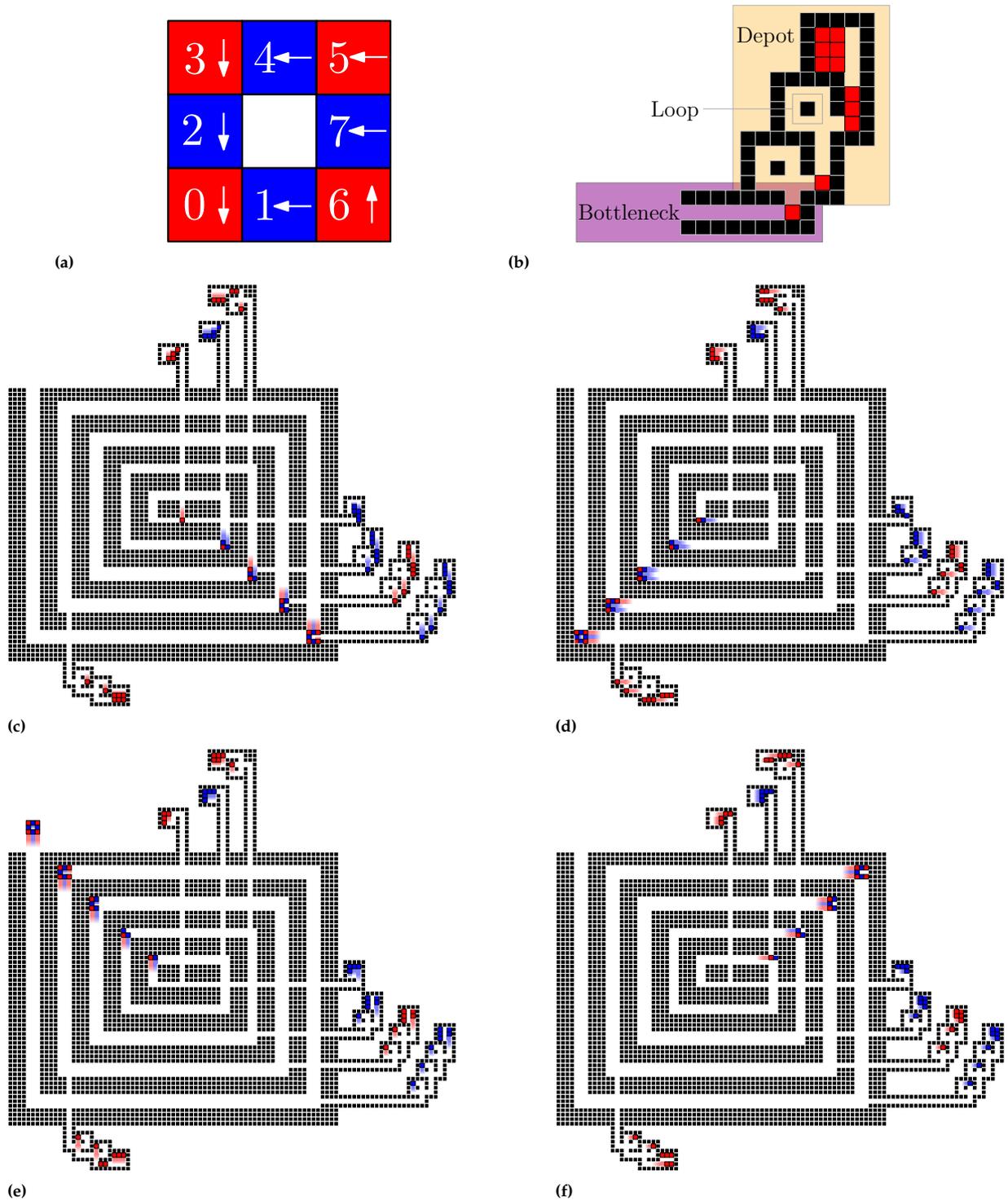


Figure 8.9.: (a) A polyomino P . Shown is the construction sequence and the direction of attachment to the seed (tile 0). (b) A depot (orange; light gray area in grayscale) having loops to delay the tile output and a bottleneck (purple; dark gray area in grayscale) to guarantee that only one tile can move to the spiral. (c to f) A maze environment for pipelined construction of the desired polyomino P . After the fourth cycle, each further cycle produces a new copy of P . Shown states are after a sequence of down (c), left (d), up (e) and right (f) moves.

proceed for each turn in the spiral as follows: For the j -th turn, if $m^{-1}(\zeta_j)$ is empty we do nothing. Else if $m^{-1}(\zeta_j)$ is not empty we want to add the next tile. Let t_i be this particular tile. Then, we construct a lane in direction $-\zeta_j$, i.e., the direction from where the tile will come from, until we are outside the spiral. By shifting this line in an orthogonal direction we can enforce the tile to fly in at the correct position relating to l_i . There, we add a depot with tiles, such that the first tile comes out after $j - 1$ steps and with each further cycle a new tile comes out (this can be done by using loops in the depot, see Figure 8.9). Depots, which lie on the same side of the spiral, can be shifted arbitrarily, so they do not collide. These depots can be made arbitrarily big, and thus, we can make as many copies of P as we wish. Note that we can make the paths in the spiral big enough, such that after every turn the bounding box of the current polyomino fits through the spiral.

Correctness: We will now show that we will obtain copies of P . Consider any j -th turn in the spiral, where the i -th tile t_i is going to be added to the current polyomino. With the next step, both t_i and the polyomino move in direction ζ_j . While the polyomino does not touch the next wall in the spiral, the distance between t_i and the polyomino will not decrease. However when the polyomino hits the wall, the polyomino stops moving and t_i continues moving toward the polyomino. Wall-hitting is the same situation as in our non-parallel model: To a fixed polyomino we can add tiles from n , e , s or w . Therefore, the tile connects to the correct place. Since this is true for any tile and any copy, we conclude that every polyomino we build is a copy of P .

Time: Since the spiral has at most $4N$ unit steps (or N cycles), the first polyomino will be constructed after $4N$ unit steps. By construction, we began the second copy one cycle after beginning the first copy, the third copy one cycle after the second, and so on. This means, after each cycle, when the first polyomino is constructed, we obtain another copy of P . Therefore, for D copies we need $N + D$ cycles (or $O(N + D)$ unit steps). For $D \in \Omega(N)$ this results in an amortized constant time construction for P .

Note that this proof only considers construction sequences in the following form:

If a tile t_i increases the side length of the bounding box of the current polyomino, then the tile is added from a direction with a longitude/latitude, such that the longitude/latitude intersects the bounding box (see Figure 8.10). In the case there is a tile, such that the longitude/latitude does not intersect the bounding box, then we can rotate the direction by $\frac{\pi}{2}$ toward the polyomino and we will have a desired construction sequence. □



Figure 8.10: Two different sequences. The red tile represents the bounding box of the current polyomino. (Left) A desired sequence. The latitude intersects the bounding box. (Right) A sequence where the latitude does not intersect the bounding box.

8.3.4. Error Detection

During the pipelined creation process, errors may occur as tiles get stuck at wrong places or do not stick at the right places. In this section, we consider *dynamic* workspaces that are composed of rigid obstacles and moving *cams* for sorting polyominoes and detecting errors. Cams are affected by the global controls in the same manner that input polyominoes are. However,

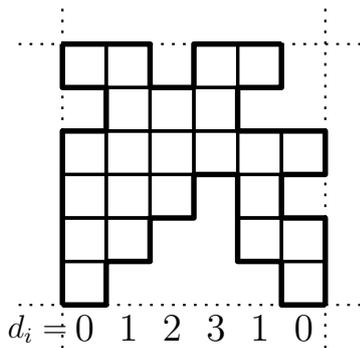


Figure 8.11: A completely filled polyomino, its lower, upper, left and right envelope (bold), the corresponding base lines (dotted) and the distance between lower base line and lower envelope.

they must not enter the input region or any output region. Moreover, we require the sorting or error reporting process to be repeatable; i.e., applying our control sequence must return the workspace to a state that can be used to sort the next incoming polyomino. Using such a workspace, we are able to identify and compare *completely filled* polyominoes, where a polyomino is called *completely filled* iff it consists of all tiles that are below its upper envelope, above its lower envelope, right of its left envelope and left of its right envelope. The *lower base line* of a completely filled polyomino is the horizontal line through its lowest points; see Fig. 8.11. Upper, left and right base lines are defined analogously.

Theorem 8.3.6 *Dynamic workspaces can sort any family \mathcal{F} of polyominoes of width up to w and height up to h that are completely filled with a sorting sequence of constant length and a workspace of dimensions $\mathcal{O}(|\mathcal{F}| \cdot w \cdot h) \times \mathcal{O}(|\mathcal{F}| \cdot w \cdot h)$.*

Proof. In the following, we describe how to construct a workspace that sorts a given family \mathcal{F} of completely filled polyominoes with a control sequence of constant length. You can get an intuition of the construction using the interactive visualization applet <https://roboticswarmcontrol.github.io/TiltSorting/index.html>. In a first step, our procedure groups the polyominoes from \mathcal{F} according to their height and width; we handle each group separately. Therefore we assume in the following that all polyominoes have the same width w and height h . Our sorting procedure checks the left, right, lower and upper envelope separately. For each envelope, constantly many operations are required; therefore, the entire procedure only requires constantly many operations. The main idea of sorting the right envelope is as follows; the construction for the other envelopes is analogous. We use a set of *pins*, one for each row of the polyomino. To sort a polyomino, the pins are pushed against the polyomino from the right. The pins consist of several stages, each stage corresponding to a certain envelope to be tested for. If the envelope matches, a set of interlocking cams called the *plug* unlocks and can be moved to the top, thereby extending a *barrier* that we then use to move the polyomino to the right position. Refer to Figure 8.12 for an example of the construction.

In the following, we describe the construction in more detail. Firstly, our construction requires a distance of three between successive rows; therefore, as a first technical step, we use one *expansion cam* per row as depicted in Figure 8.12 to introduce additional vertical space. These cams can move left and right independently of each other; therefore, they copy the right envelope of the polyomino they are pushed up against. This requires $\mathcal{O}(w \cdot h)$ space, because there must be a horizontal distance of at least w between the vertical parts of each expansion cam to allow them to move horizontally without influencing each other. To the right of the expansion cams, there is one stage for each right envelope E in \mathcal{F} . At the left end of each stage, there is a horizontal *driver* cam for each row of the polyomino. Let d_j be the distance between the right envelope and base line in row j , and let d'_j be the distance between right envelope and base line in row j in the previous stage, or 0 for the first stage. Note that due to the polyomino having width w , for at least one j we have $d_j = 0$. The driver in row j has width $3w + d_j - d'_j$. When we push all rows left

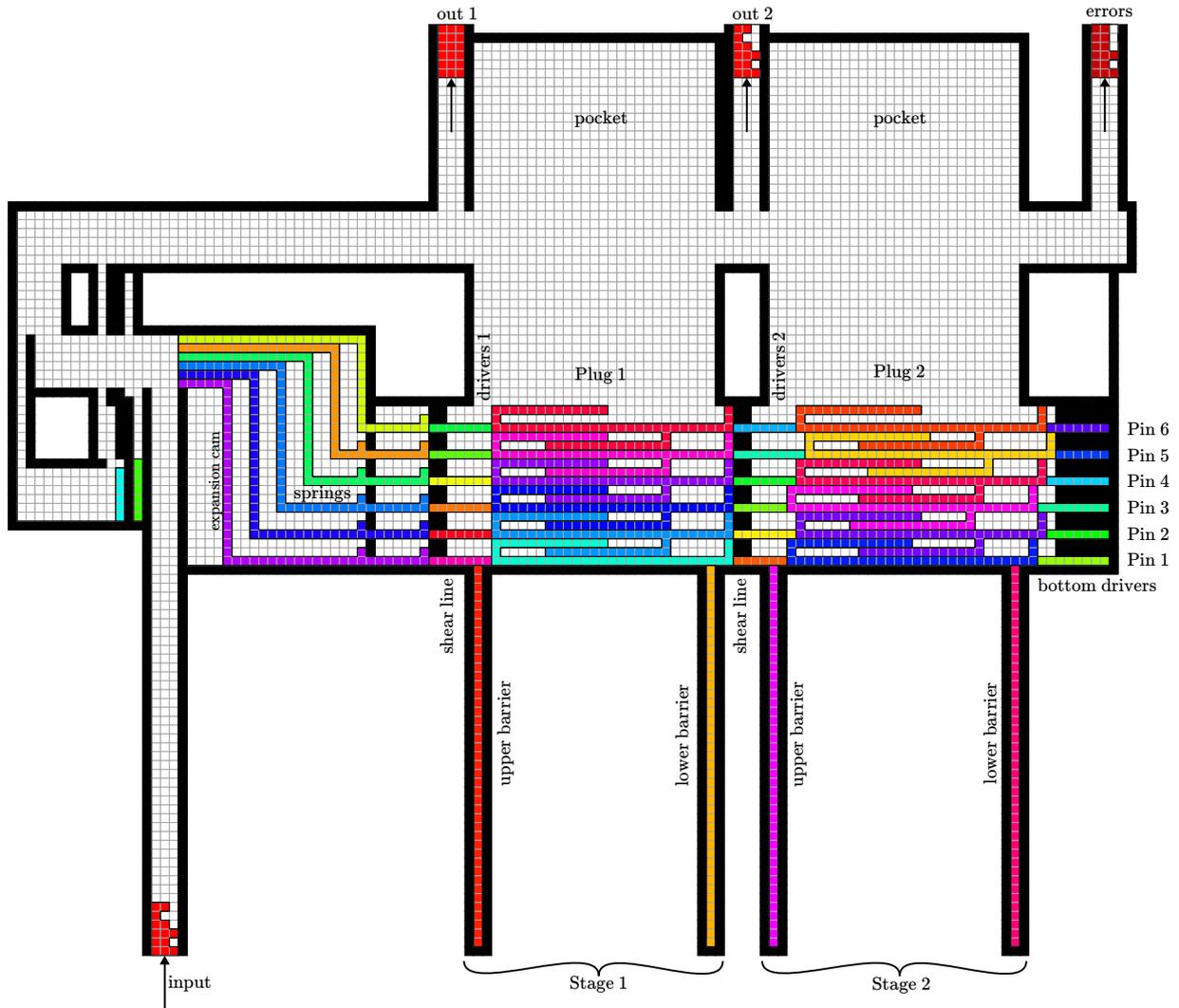


Figure 8.12.: Example of our construction that classifies polyominoes based on their right envelope. Using the control sequence $\uparrow\leftarrow\uparrow\leftarrow\uparrow\rightarrow\uparrow$ moves the red polyomino P (bottom left) out of an exit at the top depending on its right envelope. Afterwards, the sequence $\leftarrow\downarrow\rightarrow\downarrow$ resets the cams in the workspace to their initial state. The right envelope of P matches the second stage and is moved to the corresponding exit (top center); the first stage is matched by a 6×3 -rectangle, matching polyominoes leave through the first exit (top left). Any polyominoes with other envelopes leave through the last exit (top right). See <https://roboticswarmcontrol.github.io/TiltSorting/index.html> for an interactive visualization applet.

against the polyomino, this ensures that the ends of all drivers are at the same width iff its right envelope is E . We prevent any vertical motion of the drivers using rigid obstacles placed between the stages. Right of the drivers of each stage we place the *plug* of the stage. The *plug* consists of one *interlocking cam* of height 3 for each row; see Figure 8.13 for its dimensions.

The parts of each plug can move horizontally according to the right envelope of the polyomino without blocking each other. However, if one of the cams is blocked w.r.t. motion to the top, it blocks all other cams. Let y_{\top}, y_{\perp} be the leftmost and rightmost column of the plug if the polyomino has right envelope E . On the upper side of each stage, there is a horizontal wall of rigid obstacles with a window from y_{\perp} to y_{\top} . On the bottom of each stage, we add a horizontal wall of rigid obstacles with windows of width one at y_{\perp} and y_{\top} and two vertical *barriers* extending through these windows. The barriers are long vertical cams that are

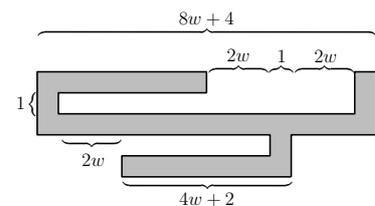


Figure 8.13.: Dimensions of interlocking cams used in our construction; each stage contains one of these cams for each row of the polyomino.

fixed at their bottom end as depicted in Figure 8.12 and cannot move horizontally; they can only move to the top if the interlocking cams are all at the same, correct width for the current stage, i.e., if the polyomino has right envelope E . In this case, the plug can move to the top into a *pocket* that prevents any motion other than to the bottom. The barriers move to the top with the plug, blocking a corridor that the polyomino travels through; their bottom end stays below the bottom wall of the stages, ensuring that the construction can be reset by a downward move. Right of the last stage, there is one more group of drivers that are held in place by narrow horizontal pockets; see Figure 8.12. \square

8.4. Optimization Variants in 2D

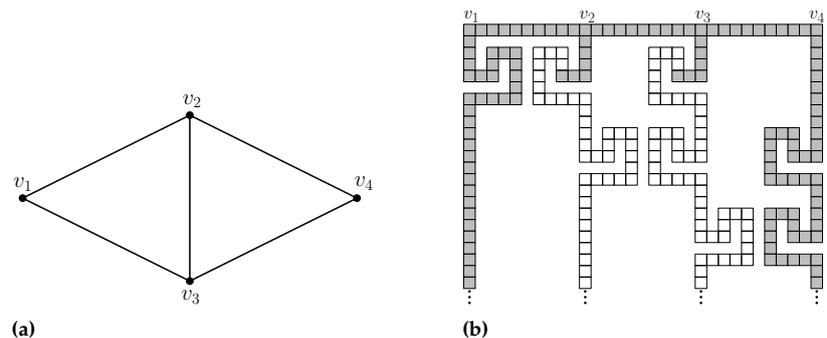
For polyominoes that cannot be assembled, it is natural to look for a maximum-size subpolyomino that is constructible. This optimization variant is *polyAPX-hard*, i.e., we cannot hope for an approximation algorithm with an approximation factor within $\Omega(N^{\frac{1}{3}})$, unless $P = NP$.

Definition 8.4.1 (Maximum Tilt Assembly Problem) *Given a polyomino P , the Maximum Tilt Assembly Problem (MAXTAP) asks for a sequence of tiles building a cardinality-maximal connected subpolyomino $P' \subseteq P$.*

Theorem 8.4.1 *MAXTAP is polyAPX-hard, even for tree-shaped polyominoes.*

Proof. We reduce MAXIMUM INDEPENDENT SET (MIS) to MAXTAP; see Figure 8.14 for an illustration. Consider an instance $G = (V, E)$ of MIS, which we transform into a polyomino P_G . We construct P_G as follows. First, construct a horizontal line from which we go down to select which vertex in G will be chosen. The line must have length $10n - 9$, where $n = |V|$. Every 10th tile will represent a vertex, starting with the first tile on the line. Let t_i be such a tile representing vertex v_i . For every v_i we add a selector gadget below t_i and for every $\{v_i, v_j\} \in \delta(v_i)$ we add a reflected selector gadget below t_j , as shown in Figure 8.14, each consisting of 19 tiles. Note that all gadgets for selecting vertex v_i are above the gadgets of v_j if $i < j$ and that there are at most n^2 such gadgets. After all gadgets have been constructed, we have already placed at most $19n^2 + 10n - 9 \leq 29n^2$ tiles. We continue with a vertical line with a length of $30n^2$ tiles.

Figure 8.14.: Reduction from MIS to MAXTAP. (Left) A graph G with four vertices. (Right) A polyomino constructed for the reduction with a feasible, maximum solution marked in gray.



Now, let α^* be an optimal solution to MIS. Then MAXTAP has a maximum polyomino of size at least $30n^2\alpha^*$ and at most $30n^2\alpha^* + 29n^2$: We take the complete vertical part of t_i for every v_i in the optimal solution of MIS. Choosing other lines block the assembly of further lines and thus, yields a smaller solution.

Now suppose we had an $N^{1-\varepsilon}$ -approximation for MAXTAP. Then we would have a solution of at least $\frac{1}{N^{1-\varepsilon}}T^*$, where T^* is the optimal solution. We know that an optimal solution has $T^* \geq 30n^2\alpha^*$ tiles and the polyomino has at most $N \leq 30n^3 + 29n^2 \leq 59n^3$ tiles. Therefore, we have at least $\frac{30n^2\alpha^*}{59^{1-\varepsilon}n^{3-3\varepsilon}}$ tiles and thus at least $\frac{1}{59^{1-\varepsilon}n^{3-3\varepsilon}}\alpha^*$ strips, because each strip is $30n^2$ tiles long. Consider some $\varepsilon \geq \frac{2}{3} + \eta$ for any $\eta > 0$, then the number of strips is $\frac{1}{59^{1/3}n^{1-3\eta}}\alpha^*$ which results in an $n^{1-\delta}$ -approximation for MIS, contradicting the inapproximability of MIS (unless P=NP) shown by Berman and Schnitger [229]. \square

As a consequence of the construction, we get Corollary 8.4.2.

Corollary 8.4.2 *Unless P = NP, MAXTAP cannot be approximated within a factor of $\Omega(N^{\frac{1}{3}})$.*

On the positive side, we can give an $O(\sqrt{N})$ -approximation algorithm for tree-shaped polyominoes.

Theorem 8.4.3 *The longest constructible path in a tree-shaped polyomino P is a \sqrt{N} -approximation for MAXTAP, and we can find such a path in polynomial time.*

Proof. Consider an optimal solution P^* and a smallest enclosing box B containing P^* . Then there must be two opposite sides of B touching at least one tile of P^* . Consider the path S between both tiles. Because (i) the area A_B of B is at least the number of tiles in P^* , (ii) $|S| \geq \sqrt{A_B}$, and (iii) a longest, constructible path in P has length at least $|S|$, we conclude that the longest constructible path is a \sqrt{N} -approximation.

To find such a path, we can search for every path between two tiles, check whether we can build this path, and take the longest, constructible path. \square

Checking constructibility for $O(N^2)$ possible paths is rather expensive. However, we can efficiently approximate the longest constructible path in a tree-shaped polyomino with the help of *sequentially* constructible paths, i.e., the initial tile is a leaf in the final path.

Theorem 8.4.4 *We can find a constructible path in a tree-shaped polyomino in $O(N^2 \log N)$ time that has a length of at least half the length of the longest constructible path.*

Proof. We only search for paths that can be built sequentially. Clearly, the longest such path is at least half as long as the longest path that can have its initial tile anywhere. We use the same search tree technique as before to look for blocking tiles. Select a tile of the polyomino as the initial tile.

Do a depth-first search and for every tile in this search, check if it can be added to the path. If it cannot be added, skip all deeper tiles, as they also cannot be added. During every step in the depth-first search, we only need to change a single tile in the search trees, doing $O(1)$ updates with $O(\log N)$ cost. As we only consider $O(N)$ vertices in the depth-first search, this results in a cost of $O(N \log N)$ for a fixed start tile. It is trivial to keep track of the longest such constructible path. Repeating this for every tile results in a running time of $O(N^2 \log N)$. \square

In tree-shaped polyominoes, finding a constructible path is easy. For simple polyominoes, additional arguments and data structures lead to a similar result.

Theorem 8.4.5 *In simple polyominoes, finding the longest of all shortest paths that are sequentially constructible takes $O(N^2 \log N)$ time.*

Before we start with the proof of Theorem 8.4.5, we show in the next two lemmas that it is sufficient to consider shortest paths only, and that we can restrict ourselves to one specific shortest path between two tiles. Hence, we just need to test a maximum of $O(n^2)$ different paths.

Lemma 8.4.6 *In a sequentially constructible path, if there is a direct straight connection for a subpath, the subpath can be replaced by the straight connection.*

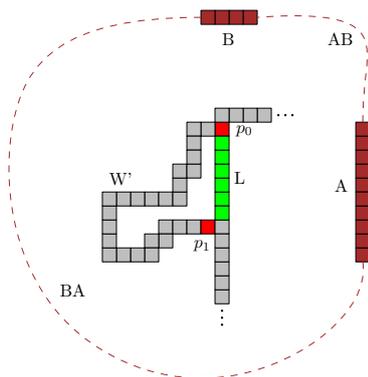


Figure 8.15. A subpath W' and its shortcut L in green. To block L , A and B must exist. But then, either p_0 or p_1 (red tiles) will also be blocked. Therefore, also W' cannot be built.

Proof. Consider a sequentially constructible path W and a subpath $W' \subset W$ that has a straight line L connecting the startpoint and the endpoint of W' . W.l.o.g., L is a vertical line and we build from bottom to top. Assume that $(W \setminus W') \cup L$ is not constructible. Then at least two structures (which can be single tiles) A and B must exist, preventing us from building L . Furthermore, these structures have to be connected via a path (AB or BA , see Figure 8.15). We observe that none of these connections can exist or otherwise, we cannot build W (if AB exist, we cannot build the last tile p_0 of L ; if BA exist, we cannot build the first tile p_1 of W'). Therefore, we can replace W' with L . \square

By repeating the construction of Lemma 8.4.6 we get a shortest path from tile t_1 to t_2 in the following form: Let P_1, \dots, P_k be reflex tiles on the path from t_1 to t_2 . Furthermore, for every $1 \leq i \leq k - 1$, the path from P_i to P_{i+1} is monotone. This property holds for every shortest path, or else we can use shortcuts as in Lemma 8.4.6.

Lemma 8.4.7 *If a shortest path between two tiles is sequentially constructible, then every shortest path between these two tiles is sequentially constructible.*

Proof. Consider a constructible shortest path W , a maximal subpath W' that is x - y -monotone, and a bounding box B around W' . Due to the L_1 -metric, any x - y -monotone path within B is as long as W' . Suppose some path within B is not constructible. Then we can use the same blocking argument as in Lemma 8.4.6 to prove that W' cannot be constructible as well, contradicting that W is constructible. \square

Using Lemma 8.4.6 and Lemma 8.4.7, we are ready to prove Theorem 8.4.5.

Proof of Theorem 8.4.5. Because it suffices to check one shortest path between two tiles, we can look at the BFS tree from each tile and then proceed like we did in Theorem 8.4.4. Thus, for each tile we perform a BFS in time $O(N)$ and a DFS with blocking look-ups in time $O(N \log N)$, which results in a total time of $O(N^2 \log N)$. \square

Future Work 8.1.

Additive production techniques, like 3D-printing, often use support structures for building otherwise unstable objects. Can we devise an approximation algorithm that makes a shape constructible with the minimum amount of additional tiles?

8.5. Three-dimensional Shapes

An interesting and natural generalization of TAP is to consider three-dimensional shapes, i.e., polycubes. The local considerations for simply connected two-dimensional shapes are no longer sufficient. In the following we show that deciding whether a polycube is constructible is NP-hard. Moreover, it is NP-hard to check whether there is a constructible path from a start cube s to an end cube t in a partial shape.

As a stepping stone, we start with a restricted version of the three-dimensional problem.

Theorem 8.5.1 *It is NP-hard to decide if a polycube can be built by inserting tiles only from above, north, east, south, and west.*

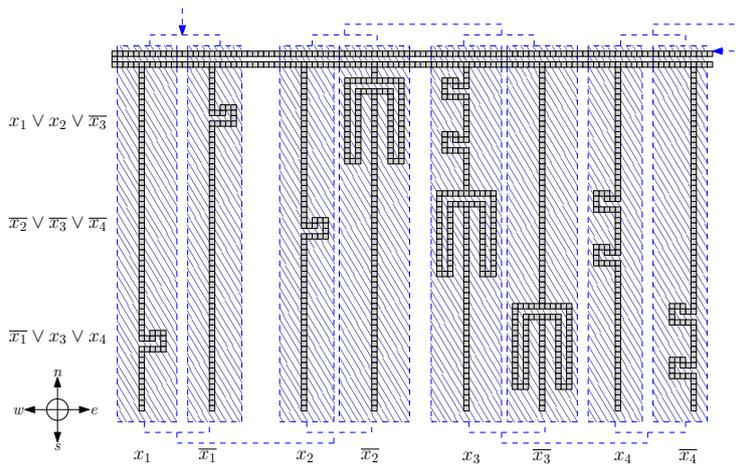


Figure 8.16.: Top-view on the polycube. There is a vertical part going south for the *true* and *false* assignment of each variable. We start building at the top layer (crosshatched area) and have to block either the *true* or the *false* part of each variable from above. The blocked parts have to be built with only inserting from east, west, and south. For each clause, the parts of the inverted literals are modified to allow at most two of them being built in this way. All other parts can simply be inserted from above in the end.

Proof. We prove hardness by a reduction from 3SAT. A visualization for the formula $(x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_2 \vee \bar{x}_3 \vee x_4) \wedge (\bar{x}_1 \vee x_3 \vee x_4)$ can be seen in Figure 8.16. It consists of two layers of interest (and some further auxiliary ones for space and forcing the seed tile by using the one-way gadget shown in Figure 8.19). Due to the one-way gadget, at least part of the top layer (crosshatched area in Figure 8.16, details in Figure 8.17)

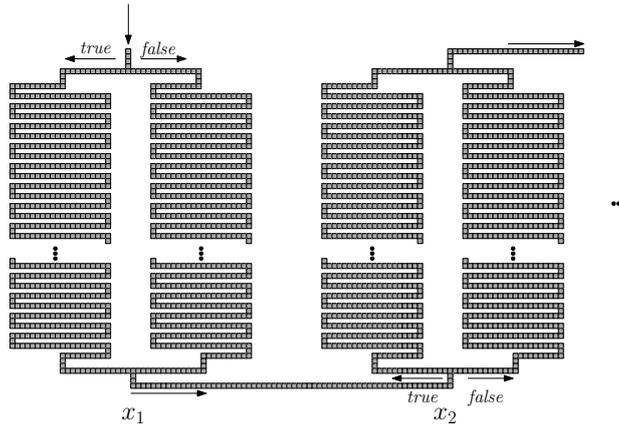


Figure 8.17.: Top-view on the polycube. In the beginning we have to block the access from the top for either the *true* or *false* part of the variable. The variable is assigned the blocked value.

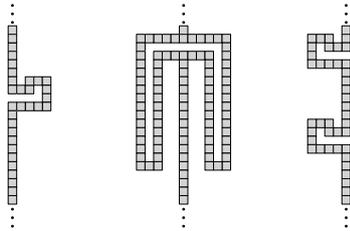
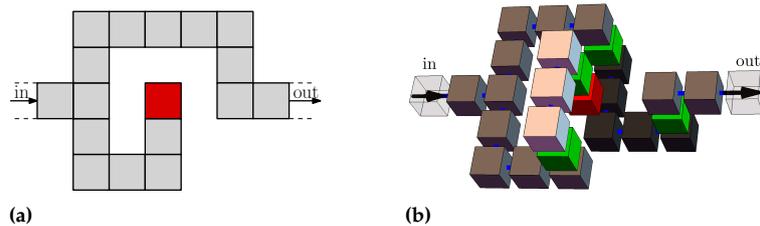


Figure 8.18.: Three gadgets for a clause. Only two of them can be built if the tiles are only able to come from the east, south, and west.

must be built first. Forcing a specific start tile can be done by a simple construction. For each variable we have to choose to block the left (for assigning *true*) or the right (for assigning *false*) part of the lower layer. In the end, the remaining parts of the upper layer can trivially be filled from above. The blocked parts of the lower layer then have to be built with only inserting tiles from east, south, or west. In the end, the non-blocked parts can be filled in from above. For each clause we use a part (as shown in Figure 8.18) that allows only at most two of its three subparts to be built from the limited insertion directions. We attach these subparts to the three variable values not satisfying the clause, i.e., the negated literals. This forces us to leave at least one negated literal of the clause unblocked, and thus at least one literal of the clause to be *true*. Overall, this allows us to build the blocked parts of the lower layers only if the blocking of the upper level corresponds to a satisfying assignment. If we can build the *true* and the *false* parts of a variable in the beginning, any truth assignment for the variable is possible.

It is straightforward to see that the whole construction fits into a bounding box of size $O(|C|) \times O(|V|) \times O(1)$, where C is the set of all clauses and V the set of all variables. \square

Figure 8.19.: (Left) This polyomino can only be constructed by starting at “in” and ending at “out”. (Right) Generalization to three dimensions. If we start on the right side, then we cannot build the red cube because it is blocked from all six directions. With these gadgets we can enforce a seed tile.



The construction can be extended to assemblies with arbitrary direction.

Theorem 8.5.2 *It is NP-complete to decide if a polycube can be built by inserting tiles from any direction.*

Proof. We add an additional layer below the construction in Theorem 8.5.1 that has to be built first and blocks access from below. Forcing the bottom layer to be built first can again be done with the one-way gadget shown in Figure 8.19. Finally, we note that the problem of deciding whether a polycube can be built by inserting tiles from any direction is in NP. \square

The difficulties of construction in 3D are highlighted by the fact that even identifying constructible connections between specific positions is NP-hard.

Theorem 8.5.3 *It is NP-complete to decide whether a path from one tile to another can be built in a general polycube.*

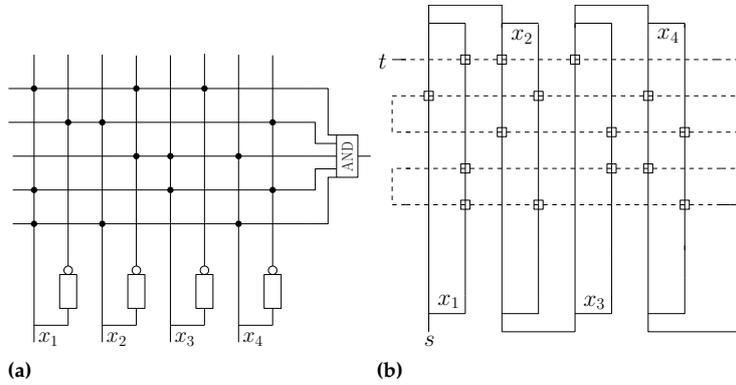


Figure 8.20.: (Left) Circuit representation for the SAT formula $(x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_4) \wedge (\bar{x}_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee x_2 \vee x_4)$. (Right) Reduction from SAT formula. Boxes represent variable boxes.

Proof. We prove NP-hardness by a reduction from SAT. For each variable we have two vertical lines, one for the *true* setting, one for the *false* setting. Each clause gets a horizontal line and is connected with a variable if it appears as literal in the clause, see Figure 8.20a. We transform this representation into a tour problem where, starting at a point *s*, one first has to go through either the *true* or *false* line of each variable and then through all clause lines, see Figure 8.20b. The clause part is only passable if the path in at least one crossing part (squares) does not cross, forcing us to satisfy at least one literal of a clause. As one has to go through all clauses, *t* is only reachable if the selected branches for the variables equal a satisfying variable assignment for the formula.

We now consider how to implement this as a polycube. The only difficult part is to allow a constructible clause path if there is a free crossing.

In Figure 8.21, we see a variable box that corresponds to the crossing of the variable path at the squares in Figure 8.20b. It blocks the core from further insertions. The clause path has to pass at least one of these variable boxes in order to reach the other side. See Figure 8.22 for an example. Note that the corresponding clause parts can be built by inserting only from above and below, so there are no interferences. □

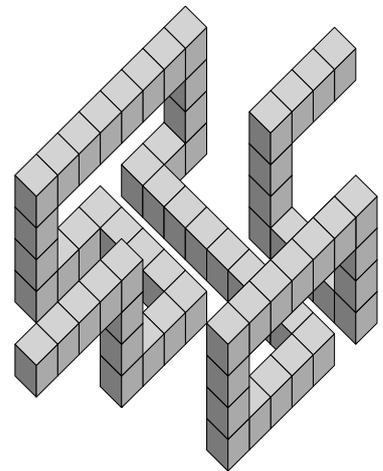


Figure 8.21.: Empty variable box.

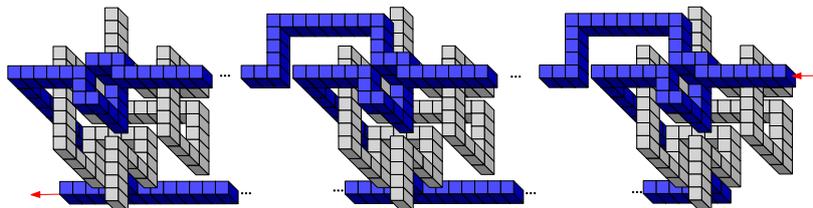


Figure 8.22.: A clause line (blue dark gray in grayscale) dips into a variable box. If the variable box is built, then we cannot build the dip of the clause line.

Future Work 8.2.

Can we decide in polynomial time if simple types of polycubes can be constructed, e.g., polycubes that can be iteratively cut into exactly

two connected components by straight cuts?

8.6. Computational Study

In this section, we implement a solver which can compute a feasible construction sequence, or prove non-constructability, for arbitrary polyominoes and polycubes. We know that deciding constructability for polycubes is NP-complete, so we cannot expect an efficient algorithm. Of course, we can easily devise construction heuristics, but to prove that something is not possible is much harder. The naïve approach is to perform an exhaustive search of all possible construction sequences. This will fail even for relatively small instances as there are simply too many possibilities. Backtracking approaches can be much more efficient, so let us first take a look at such an approach and explain its limitations.

The general idea of backtracking is to traverse the decision tree and try to detect conflicts as early as possible (and not just at the end of the decision tree). An example of such a conflict is when an empty position of the polyomino/polycube gets cut off by the already-constructed part and cannot be filled anymore. This can be detected very quickly since in each step, only a small part of the instance can be influenced. When we detect such a conflict, we try to jump back as far as possible to the point where the conflict originated, as all further decisions are inherently doomed to failure. If a polyomino is not constructible, the conflict originates from the beginning, but this is hard to detect. In Figure 8.23 we see a backtracking algorithm which just placed the 24th tile and detects a conflict: the tile with the red '?' becomes unreachable. We can trivially jump back to the 20th tile, but we have an exponential number of possibilities to reach it. Especially the blue part can be heavily varied before we have exhausted all options. Thus, we need a more refined way to detect earlier that this polyomino cannot be built. As the polyomino is actually hole-free, this is trivially possible (Theorem 8.3.4), but for a polyomino with holes, this is far more difficult.

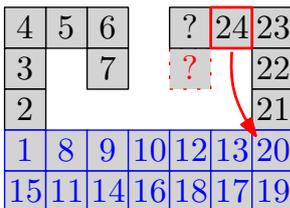


Figure 8.23.: Backtracking in this non-constructible polyomino is difficult.

Instead of trying to find a set of clever heuristics for detecting conflicts, which involves a lot of theoretical work and practical engineering, we use already existing generic conflict detection techniques by transforming the problem into a SAT-formula and a constraint program. We first describe how to formulate our problem as a SAT-problem in Subsection 8.6.1. Afterward, we formulate it as a constraint program which allows more powerful expressions, in Subsection 8.6.2, but is also based on SAT (using CP-SAT). A computational evaluation of the practicality of these approaches is then performed in Subsection 8.6.3. A description of how the corresponding solvers work is provided in Section 3.4.3 (How Does CP-SAT Work?) on page 65.

8.6.1. Solving as SAT-Formula

We now discuss how we can express the constructability of a polyomino or polycube as a boolean formula, i.e., a SAT-problem. The most significant decision we have to make is ‘which tile do we place before which?’. For any two tiles $t, t' \in P$, let $X_{t < t'} = \overline{X_{t' < t}} \in \mathbb{B}$ denote that t is built before t' .

We can actually deduce everything else from these variables but to make things easier to read and clauses shorter, let us introduce some further variables.

We denote with $S_t \in \mathbb{B}$ if $t \in P$ is the first tile. We can only have one first tile, thus, we have to add the clauses

$$\forall t \neq t' \in P : S_t \Rightarrow \overline{S_{t'}} \quad (8.1)$$

If the SAT-solver supports native cardinality constraints, we can also state $\sum_{t \in P} S_t = 1$.

Every tile $t \in P$ that is not built first, needs to be built from one direction and have an adjacent tile $t' \in N(t)$ to dock to, i.e., $X_{t' < t} = \text{true}$. Let D_t^o denote that t is built from direction o and O be the set of possible directions (n, e, s , and w for polyominoes; polycubes additionally have top and bottom). This results in

$$\forall t \in P : S_t \vee \bigvee_{o \in O} D_t^o \quad (8.2)$$

and

$$\forall t \in P : \overline{S_t} \Rightarrow \bigvee_{t' \in N(t)} X_{t' < t}. \quad (8.3)$$

We can also transform the first constraint into a cardinality constraint. This is actually redundant, but in some cases such redundancies can drastically speed up the solver.

If tile $t \in P$ is built from direction $o \in O$, all blocking tiles in that direction of course have to be built later. Let $\mathcal{B}_o(t) \subseteq P$ denote this blocking set.

$$\forall t \in P, o \in O, t' \in \mathcal{B}_o(t) : D_t^o \Rightarrow X_{t < t'} \quad (8.4)$$

This already describes our problem pretty well: we enforce that a tile has something to dock to when built, and that nothing blocks it. However, the order, i.e., the X -variables, can still be inconsistent, i.e., contain cycles. To prevent this, we have to enforce transitivity. This can easily be done by

$$\forall t_0 \neq t_1 \neq t_2 \in P : X_{t_0 < t_1} \wedge X_{t_1 < t_2} \Rightarrow X_{t_0 < t_2}. \quad (8.5)$$

This creates a cubic number of clauses which does not scale very well.

To reduce the number of clauses, we can also add them lazily whenever the returned solution violates one. Many SAT-solver support iterative construction and solving of formulas, i.e., variables and clauses can be added to a solved formula. The clauses and weights already learned are reused, thus, it continues where it stopped (comparable to the commonly used callback in a MIP-solver).

For a lazy construction, we start with all constraints except the ones in Equation 8.5, and solve the formula. Additionally, some X -variables, which do not appear in any clauses, are left out. If the formula is infeasible, we can stop searching, because adding clauses cannot make the model feasible again. If, on the other hand, we have an assignment, we need to look for violations of the constraints in Equation 8.5. For this, we first check if there is some $X_{t_0 < t_1} \wedge X_{t_1 < t_2} \wedge \overline{X_{t_0 < t_2}}$ and directly add a corresponding clause preventing this assignment. This check can be

executed quickly using appropriate data structures. Because not all X -variables are already contained, we also need to look for larger cycles. This is more costly and, thus, we only need to perform this action if we have not found any conflicts beforehand. For a larger cycle, we add all transitivity constraints for all triples within it.

8.6.2. Constraint Programming

We saw that we may need a cubic number of constraints when encoding the ordering of the assembly sequence as a boolean relationship. Constraint programming allows us to encode this ordering as integral values that are already transitive by design. Let $I_t \in \mathbb{N}_0^{|P|-1}$ describe the position of tile $t \in P$ in the assembly sequence. To make this assignment unique, we only need to add the single constraint

$$\text{ALLDIFFERENT}(I_t \mid t \in P). \quad (8.6)$$

We reuse the D - and S -variables from the SAT-formulation. To make sure every tile (except the first tile) has a direction, we can add the constraint

$$\forall t \in P : \sum_{o \in O} D_t^o = 1 - S_t. \quad (8.7)$$

In this case, we again have to make sure that the tile can actually dock to something.

$$\forall t \in P : \min_{t' \in N(t)} I_{t'} \leq I_t - 1 \text{ if } \overline{S_t} \quad (8.8)$$

And finally, there should be no blocking tiles.

$$\forall t \in P, o \in O : \min_{t' \in \mathcal{B}_o(t)} I_{t'} \geq I_t + 1 \text{ if } D_t^o \quad (8.9)$$

This formulation needs significantly fewer constraints (linear instead of cubic) and variables (linear instead of quadratic) than the SAT-formulation. This does not necessarily imply that it is also faster, because the constraints are more complex. Also, the employed solver CP-SAT implicitly converts the variables into boolean variables. The used encoding, as explained in Section 3.4.3 (How Does CP-SAT Work?) on page 65, also uses a quadratic number of variables in the worst case (lazy creation).

8.6.3. Evaluation

In this section, we evaluate the performance of the SAT-formulation with multiple SAT-solvers, and the performance of the CP-formulation with CP-SAT of Google's ortools [70]. Before we can run any experiments though, we first need to create some instances. This is not trivial because most random instances are constructible, and furthermore are often constructible using simple heuristics. To have the best chance of creating non-constructible instances, we start with a random variant of the spiral in Figure 8.2. Then we randomly select a tile and extend it in a random direction. To make the instance more ramified, we can add multiple tiles in that same direction. By making the selection probability of a tile linearly dependent on the number of free sides, we increase the chance of

creating spirals further. A sample of the resulting instances can be seen in Figure 8.28.

We compare the following SAT-solver packaged by PySat [230] with a uniform interface:

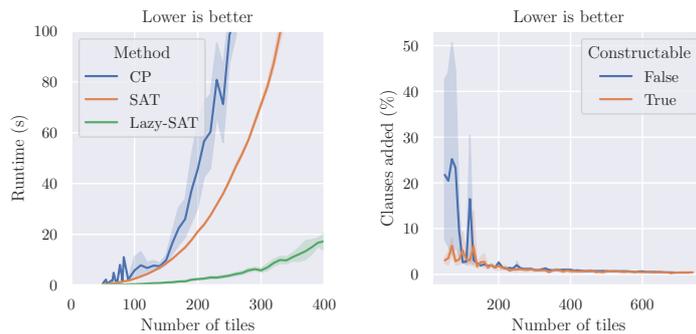
- ▶ Glucose (3.0 and 4.1) [231]
- ▶ Maplesat (MapleCOMSPS_LRB) [232]
- ▶ Mergesat (3.0) [233]
- ▶ Minisat (2.2 release and GitHub version) [234]

While some of the solvers support native cardinality constraints or have variants that do, we leave this part to future work.

Future Work 8.3.

Can we improve the performance further by using native cardinality constraints or adding fewer constraints when using lazy constraints?

Let us first decide which formulation and approach is the fastest. Since we have a set of SAT-solvers, we always select the best solver for the corresponding instance. The SAT-solvers themselves are compared later. The plot in Figure 8.24a shows an inferior performance of constraint programming and SAT with all clauses added from the beginning. They



(a) Runtime of different methods.

(b) Relative size of lazy formulation.

Figure 8.24.: Runtime comparison of instance size for CP and SAT. For the SAT-variants, the SAT-solver with the lowest runtime for this instance has been chosen. The median datapoint consists of 30 instances. The SAT-solvers with lazy clause generation perform best as they only add a fraction of the clauses of the full formulation.

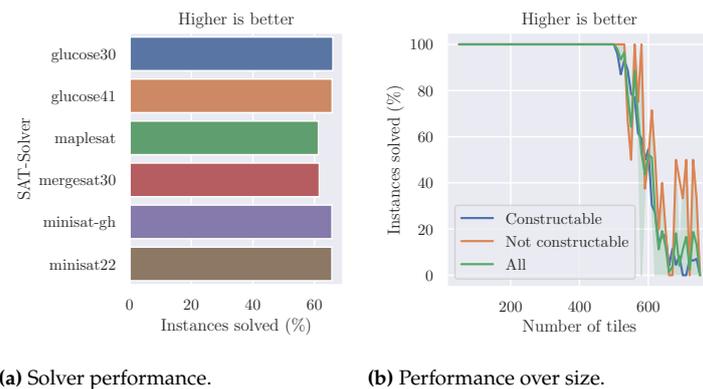
can still solve instances with over 200 tiles reliably but the runtime increases quickly. It is important to note that solving the large SAT-formulas is not the bottleneck; creating the cubic sized formula takes up most of the time. Creating the formulas via optimized native code would decrease the runtime drastically. However, the lazy clause generation is clearly the fastest method. It needs quite a number of iterations and lazy clauses for some instances. The lazy clauses are actually slower to create than for the full version because we first have to search for them, but we still end up with far fewer clauses. This can be seen in Figure 8.24b showing how many of the original clauses are actually added, which on average is less than 2.4%. Surprisingly, small non-constructible instances need a much higher percentage despite the fact that we can abort for such an instance as soon as we have added enough clauses to make the underlying SAT-formula unsatisfiable. Missing transitivity constraints seem to allow the assignment to evade unsatisfiability to quite some depth. A reason why this only happens for small instances may be that the unsatisfiable spiral comprises a larger fraction of a small instance than

for a large instances. Thus, using a SAT-solver with lazy clauses is the most efficient solution for this problem.

The SAT-solvers perform nearly equally well for lazy clause generation, as can be seen in Figure 8.25a. Of the instance set with 1722 instances with up to 750 tiles each, the solvers were able to solve roughly 66 % within 60 seconds. Only *maplesat* and *mergesat* performed worse and solved only 63 %. The *minisat*-solver (GitHub-version) actually performed best while being reasonably simple. The advanced features of the other solvers, building upon *minisat*, did not yield more instances solved. This indicates that the corresponding formulas are reasonably easy to solve or disprove, and that the size is the primary problem.

The implementation could solve instances with around 500 tiles within the time limit, regardless of whether they are constructible or not. This can be seen in Figure 8.25b, showing how many instances of which size could be solved within the time limit by any of the SAT-solvers. The data for the non-constructible instances is relatively sparse because only 355 of the 1722 instances were non-constructible. Also, the way they were created possibly made them simpler. Hence, we should be careful with these observations as they may be biased.

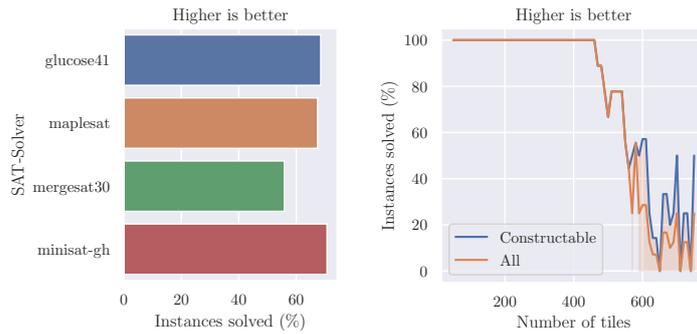
Figure 8.25.: The first plot shows how many instances have been solved within 60 s using the corresponding SAT-solver with lazy clause generation. The SAT-solvers performed nearly equally well, except for *maplesat* and *mergesat*, which are slightly weaker. The second plot shows how many instances of which size have been solved by any of the SAT-solvers, split for constructability. At around 500 tiles, instances begin to become unsolvable in time for all types.



For two-dimensional instances, it remains to be seen whether non-simple polyominoes are NP-hard. For three-dimensional instances, we already know that they are hard. Thus, it is interesting to evaluate the performance of the approach for three-dimensional instances as well. The instances were created in a similar way as for 2D, except that we do not start with a spiral. The test bed contained over 600 instances of uniformly distributed size between 50 and 750 tiles. We can see in Figure 8.26b that the solvable instance size has decreased slightly but remains above 500. This decrease can be explained by the slightly larger formulation. The SAT-solver *minisat* remains the best solver, as shown in Figure 8.26a. All solved instances are constructible. Due to the fact that most of the instances could not be solved in time were large instance, it is not to be expected that they contain many non-constructible instances. While the extra dimension makes the problem harder to decide in specific cases, they make random instances easier to construct.

Future Work 8.4.

How can we create harder instances, especially for 3D? We do not yet



(a) Solver performance.

(b) Performance over size.

Figure 8.26.: Results for three-dimensional instances. We can solve nearly similar-sized instances as for 2D, but we were not able to create any non-constructible instances. The lines for *constructible* and *all* differ because we let the solver run longer than the time limit in the hope of finding out if the instance is constructible. Instances with a duration of longer than 60s have still been marked as “not solved”.

know which shape practical instances would be, so harder does not necessarily mean more realistic. It is still useful to perform this work and be prepared, in case realistic instances are indeed hard.

Future Work 8.5.

Can we engineer a solver which approximates the desired shape but always finds an assembly sequence? Approximation can be performed either like MAXTAP or by adding as few additional tiles as possible. A combination of both is also imaginable. Such an optimization is much harder to express as a SATISFIABILITY problem. For this, the native cardinality constraints may be useful to perform an iterative search on the number of missing or additional tiles. This can also be implemented directly with a MAX-SAT-solver which is able to differentiate between hard and soft constraints.

8.7. Conclusion

We have provided a number of algorithmic results for Tilt Assembly, and we have engineered a solver based on SAT. But various unsolved challenges remain. What is the complexity of deciding TAP for non-simple polyominoes? While Lemma 8.3.3 can be applied to all polyominoes, we cannot simply remove any locally convex tile. Can we find a constructible path in a general polyomino from given start- and endpoints? This would help in finding a \sqrt{N} -approximation for non-simple polyominoes. How can we optimize the total makespan for constructing a shape? And what options exist for non-constructible shapes?

An interesting approach is to consider a *staged* assembly, as shown in Figure 8.27, where a shape gets constructed by putting together subpolyominoes, instead of adding one tile at a time. This is similar to staged tile self-assembly [207–209] and provides a path to sublinear assembly times, as a hierarchical assembly allows massive parallelization. Based on this approach, Schmidt et al. [235] are able to achieve sublinear construction time for a various types of polyominoes. However, much future work remains for general polyominoes and polycubes.

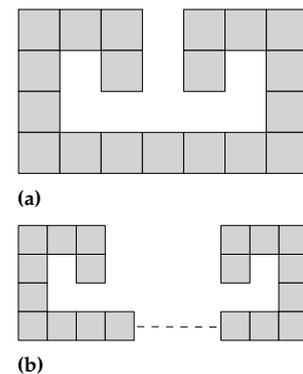


Figure 8.27.: (Top) A polyomino that cannot be constructed in the basic TAP model. (Bottom) Construction in a staged assembly model by putting together subpolyominoes.

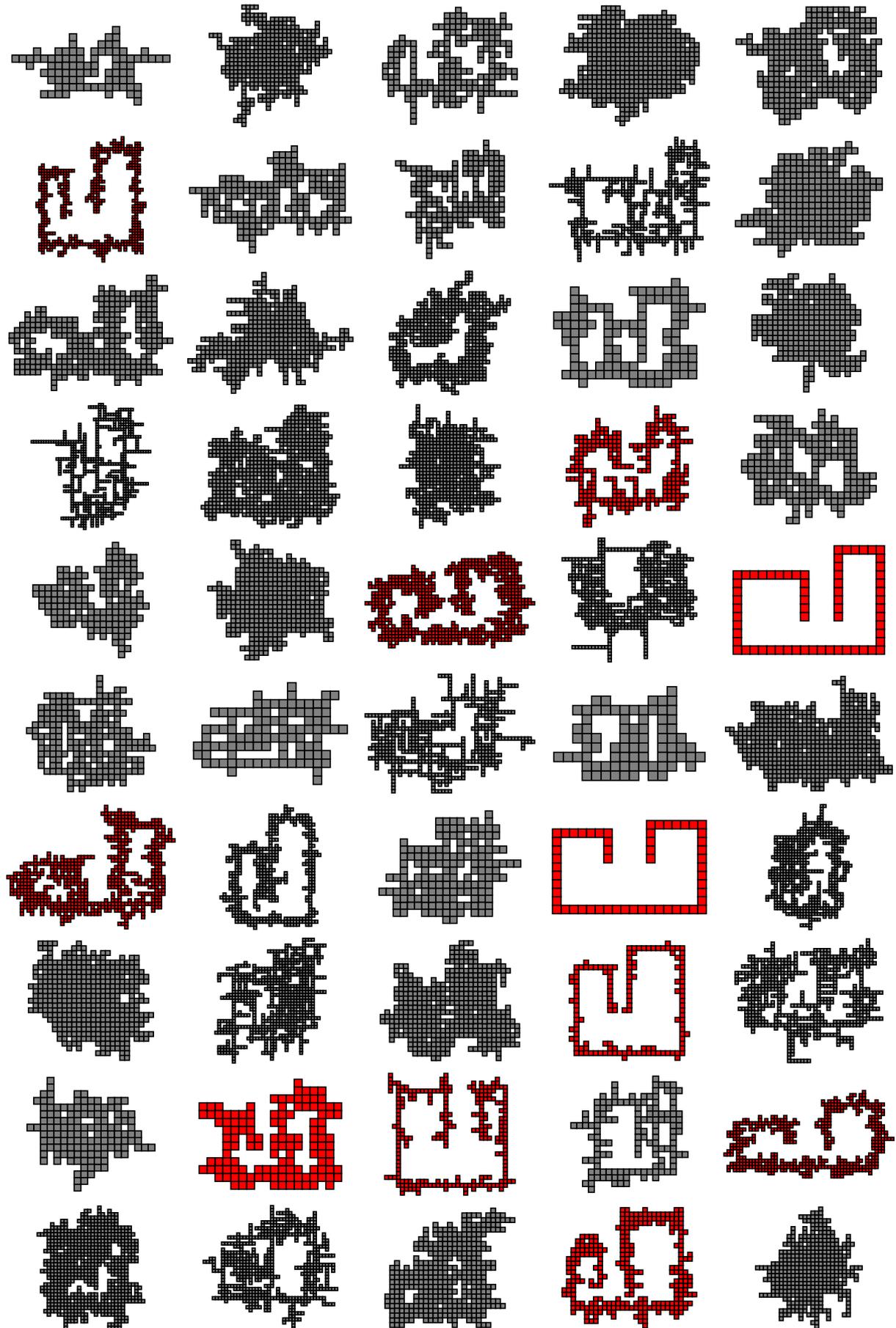


Figure 8.28.: Examples of the random instances for 2D. Instances in red are not constructible.

Targeted Drug Delivery*

9.

This chapter investigates algorithmic approaches for targeted drug delivery in a complex, maze-like environment, such as the human vascular system. The basic scenario is given by a large swarm of micro-scale particles (“agents”) and a particular target region (“tumor”) within a system of passageways. Agents are too small to contain on-board power or computation and are instead controlled by a global external force that acts uniformly on all particles, such as an applied fluidic flow or electromagnetic field. The challenge is to deliver all agents to the target region with a minimum number of actuation steps. We show hardness of the underlying problem, provide algorithms with performance guarantees, and computationally evaluate them as well as heuristics based on local search and deep reinforcement learning. The approach based on deep reinforcement learning significantly outperforms the classical approaches in these experiments.

9.1 Introduction	203
9.1.1 Overview	204
9.1.2 Related Work	204
9.2 Preliminaries	205
9.3 Algorithmic Approaches	206
9.3.1 Complexity	206
9.3.2 Merging Two Particles	208
9.3.3 Reducing the Number of Particles	211
9.3.4 General Upper Bounds	212
9.4 Reinforcement Learning	212
9.4.1 Reward	214
9.4.2 Implementation	216
9.5 Evaluation	217
9.5.1 Oblivious Merging	220
9.6 Conclusions	220

9.1. Introduction

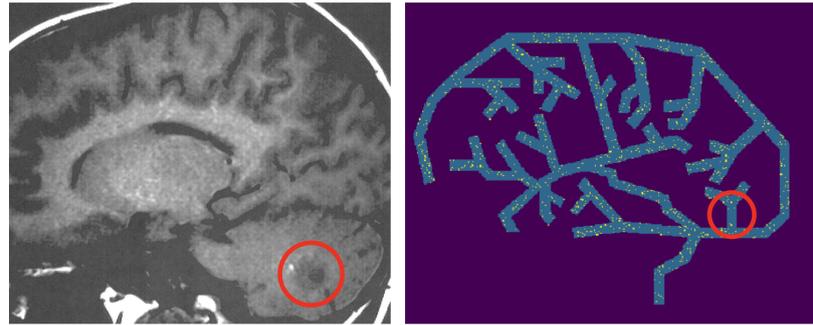
A crucial challenge for a wide range of vital medical problems, such as the treatment of cancer, localized infections and inflammation, or internal bleeding is to deliver active substances to a specific location in an organism. The traditional approach of administering a sufficiently large supply of these substances into the circulating blood may cause serious side effects, as the outcome intended for the target site may also occur in other places, with often undesired, serious consequences. Moreover, novel custom-made substances that are specifically designed for precise effects are usually in too short supply to be generously poured into the blood stream. In the context of targeting brain tumors (see Figure 9.1), an additional difficulty is the blood-brain barrier. This makes it necessary to develop other, more focused methods for delivering agents to specific target regions.

Given the main scenario of medical applications, this requires dealing with navigation through complex vascular systems, in which access to a target location is provided by pathways (in the form of blood vessels) through a maze of obstacles. However, the microscopic size of particles necessary for passage through these vessels makes it prohibitively difficult to store sufficient energy in suitably sized microrobots, in particular in the presence of flowing blood.

A promising alternative is offered by employing a global external force, e.g., a fluidic flow or an electromagnetic field. When such a force is applied, all particles move in the same direction by the distance, unless they are blocked by obstacles in their way. While this makes it possible to move all particles at once, it introduces the difficulty of using *uniform* forces for many particles in *different* locations with different local topology

* This chapter is based on [10] with full proofs. The reinforcement learning approach has been designed from scratch, and the experimental evaluation has been updated.

Figure 9.1.: (Left) An MRI image of a brain tumor (marked by the red circle), located in the cerebellum. (Right) How can the swarm of particles (indicated by yellow dots) be delivered to the target region?



to navigate them to *one* final destination. In this chapter, we investigate how this objective can be achieved with a small number of actuator steps.

Previous work [22] described a basic approach that delivers all particles in a grid environment with n grid cells to a target in at most $O(n^3)$ actuator steps. This shows that delivery can always be achieved; however, a delivery time of this magnitude is usually impractical, which is why we investigate possible improvements.

9.1.1. Overview

We discuss a number of insights:

- ▶ We prove that minimizing the length of a command sequence for gathering all particles is NP-hard, even for environments that consist of grid cells in the plane, so no polynomial-time algorithms can be expected. This explains the observed difficulty of the problem and also implies hardness for the related localization problem.
- ▶ We develop an algorithmic strategy for gathering all particles with a worst-case guarantee of at most $O(kD^2)$ steps; here D denotes the maximum distance between any two points of the environment and k the number of its convex corners. Both k and D are usually much smaller than the number n of grid locations in the environment: n may be in $\Omega(D^2)$, for two-dimensional and in $\Omega(D^3)$ for three-dimensional environments.
- ▶ For the special case of hole-free environments, we can gather all particles in $O(kD)$ steps.
- ▶ We successfully apply deep learning to search for short command sequences in individual, complex instances.
- ▶ We perform a simulation study of the various approaches, evaluating the respective performance for application-inspired instances.

9.1.2. Related Work

This chapter seeks to understand control for large numbers of microrobots, and uses a generalized model that could apply to a variety of drug-carrying microparticles. An example are particles with a magnetic core and a catalytic surface for carrying medicinal payloads [236, 237]. An alternative are aggregates of *superparamagnetic iron oxide microparticles*, $9 \mu\text{m}$ particles that are used as a contrast agent in MRI studies [238].

Real-time MRI scanning can allow feedback control using the location of a swarm of these particles.

Steering magnetic particles using the magnetic gradient coils in an MRI scanner was implemented in [236, 239]. 3D Maxwell-Helmholtz coils are often used for precise magnetic field control [238]. Still needed are motion planning algorithms to guide the swarms of robots through vascular networks. To this end, we build on the techniques for controlling many simple robots with uniform control inputs presented in [201, 214, 240]; see video and abstract [241] for a visualizing overview. For a recent survey on challenges related to controlling multiple microrobots (less than 64 robots at a time), see [242]. Further related work includes assembling shapes by global control (e.g., see [8, 243]) or rearranging particles in a rectangle of agents in a confined workspace [244, 245].

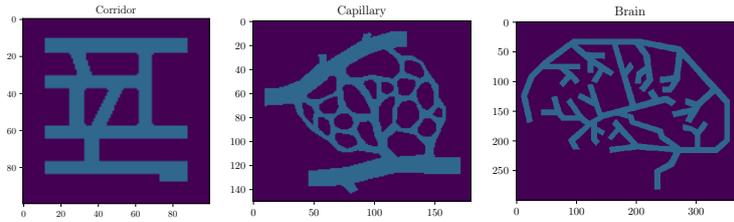
As the underlying problem consists of bringing together a number of agents in one location, a highly relevant algorithmic line of research considers *rendezvous search*, which requires two or more independent, intelligent agents to meet. Alpern and Gal [246] introduced a wide range of models and methods for this concept as have Anderson and Fekete [247] in a two-dimensional geometric setting. Key assumptions include a bounded topological environment and robots with limited onboard computation. This is relevant to maneuvering particles through worlds with obstacles and implementation of strategies to reduce computational burden while calculating distances in complex worlds [248]. In a setting with autonomous robots, these can move independent of each other, i.e., follow different movement protocols, called *asymmetric rendezvous* in the mathematical literature [246]. If the agents are required to follow the same protocol, this is called *symmetric rendezvous*. This corresponds to our model in which particles are bound by the uniform motion constraint; symmetry is broken only by interaction with the obstacles. For an overview of a variety of other algorithmic results on gathering a swarm of autonomous robots, see the recent survey by Flocchini [249]; note that these results assume a high degree of autonomy and computational power for each individual agent, so their applicability for our scenarios is quite limited.

9.2. Preliminaries

The “robots” in this chapter are simple particles without autonomy. We assume that their size is insignificant compared to the elementary cells in the workspace P . For simplicity, our description focuses on planar workspaces P , consisting of orthogonal sets of cells, so-called *pixels*, that form an edge-to-edge connected domain in the integer planar grid, i.e., a *polyomino*. (As we sketch in appropriate places, an extension to three-dimensional workspaces is largely straightforward.) An example of a polyomino is illustrated in Figure 9.3. Pixels in the planar grid not belonging to P are *blocked*: They form obstacles for particles that stop the motion from an adjacent pixel.

The particles are commanded in unison: In each step, all particles are relocated by one unit in one of the directions “Up” (u), “Down” (d), “Left” (l), or “Right” (r), unless the destination is a blocked pixel; in this case,

Figure 9.2.: The polyominoes Corridor, Capillary, and Brain on which we evaluate the approaches.



a particle remains in its previous pixel. A motion plan is a command sequence $C = \langle c_1, c_2, c_3, \dots \rangle$, where each command $c_i \in \{u, d, l, r\}$. For a command sequence C and a non-negative integer ℓ , we denote the command sequence consisting of ℓ repetitions of C by C^ℓ .

Because the particles are small, many of them can be located in the same pixel. During the course of a command sequence, two particles π_1 and π_2 may end up in the same pixel p , if π_1 moves into p , while π_2 remains in p due to a blocked pixel. Once two particles share a pixel, any subsequent command will relocate them in unison—they will not be separated, so they can be considered to be *merged*.

The distance $\text{dist}(p, q)$ between two pixels p and q is the length of a shortest path on the integer grid between p and q that stays within P . The *diameter* of a polyomino P describes the maximum distance between any two of its pixels; we denote it by D .

A *configuration* of P is a set of pixels containing at least one particle. The set of all possible configurations of P is denoted by \mathcal{P} . We call a command sequence *gathering* if it transforms a configuration $A \in \mathcal{P}$ into a configuration A' such that $|A'| = 1$, i.e., if it merges all particles in the same pixel. Gathered particles can easily be directed to any target via a shortest path.

For evaluation, we use three different workspaces: Corridor, Capillary, and Brain. These are shown in Figure 9.2.

9.3. Algorithmic Approaches

In this section, we investigate several algorithmic approaches for two-dimensional scenarios. We start by showing that the problem is computationally hard – for several variants.

9.3.1. Complexity

We show that the following decision problem, which we call MIN-GATHERING, is hard: Given a polyomino P and a set of particles, is there a gathering sequence of length ℓ ?

Theorem 9.3.1 *MIN-GATHERING is NP-hard, even for the case of polyominoes.*

Proof. We reduce from 3-SAT, i.e., the problem of deciding of whether a given boolean formula has a truth assignment; for more background see [250]. For every instance Φ of 3-SAT, we construct a polyomino P_Φ as

follows: For every variable, we insert a variable gadget as indicated in Figure 9.3. We join all variable gadgets vertically in row to a *variable block*; we call the top row of each variable gadget its *variable row*. For every clause, we construct a clause gadget that contains a left (right) arm for each incident positive (negative) literal in the corresponding variable row and an exit arm in the bottom. To obtain P_Φ , we join all clause gadgets from left to right by a *bottom row* and insert a variable block at the left and right end of the bottom row. For an illustration, consider Figure 9.3.

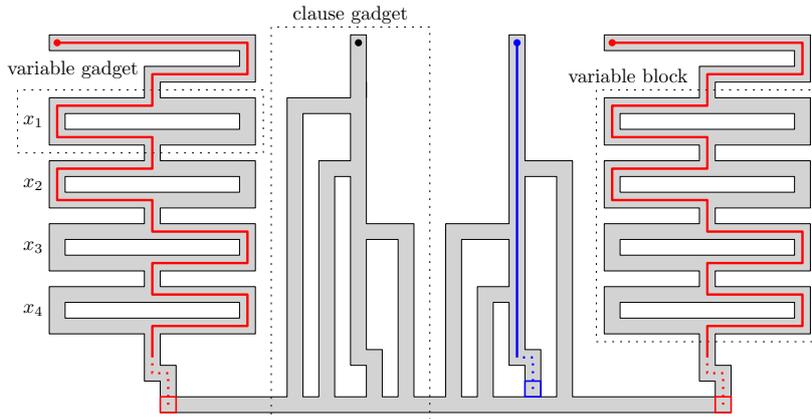


Figure 9.3: The polyomino P_Φ for the 3-SAT-instance $\Phi = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_2 \vee x_3 \vee x_4)$. A sequence that merges the two red particles with $\frac{1}{2}(D + b)$ commands corresponds to a variable assignment of Φ .

Let I be the instance of MIN-GATHERING consisting of P_Φ where the top row is filled with particles. We call the two leftmost particles above the variable blocks, the *red* particles and denote the length of the bottom row by b . Note that the distance between the red particles is the diameter D .

Claim 9.3.2 *I has a gathering sequence of length $\ell := \frac{1}{2}(D + b)$ if and only if Φ is satisfiable.*

If Φ is satisfiable, consider a satisfying assignment of Φ and apply the command sequence that moves the left red particle to the left pixel of the bottom row such that it moves left in the variable row of x_i if x_i is true and right, otherwise. Note that each particle of a clause gadget uses an arm of a variable satisfying the clause and thus ends in the bottom row of P_Φ . Then the command sequence $\langle l \rangle^b$ merges all particles. This gathering sequence has length ℓ .

Now we consider the case that Φ is not satisfiable. We show that in a gathering sequence of length ℓ , the red particles must merge in the bottom row: They do not merge in one of the variable blocks, otherwise the distance of one particle to the merge location exceeds ℓ . Moreover, the two particles move symmetrically through the variable blocks for (at least) the first $\ell - b$ commands. Reaching the bottom row of length b on opposite ends, they can only merge within b steps at the left or right end of the bottom row.

Consequently, the gathering sequence is determined by merging the two red particles; only the choice of going left or right in each variable gadget is to be determined, yielding a one-to-one correspondence to each variable assignment. Because Φ is unsatisfiable, in every variable assignment there exists a clause that is not satisfied. Consider the top particle of this clause, which we call the *blue particle*. When traversing the clause gadget, the blue particle does not use any variable arm while

the red particle traverses the variable block; because the exit arms of the variable blocks and clause gadgets are on different heights, the blue particle ends one pixel before the bottom row after $\ell - b$ steps. Thus, the red and the blue particles cannot be merged by b commands. \square

Note that the left pixel of the bottom row is one of two possible merge location for a gathering sequence of length $\frac{1}{2}(D + b)$. Therefore, the same reduction shows that problem remains hard if a target location is prescribed. In fact, an even stronger statement holds true: An instance of the polyomino P_Φ where all pixels are filled has a gathering sequence of length $\frac{1}{2}(D - b)$ if and only if Φ is satisfiable. This implies that the decision problem of ROBOT LOCALIZATION is also hard. In an instance of this problem, we are given a sensorless robot r in a polyomino, and wonder whether there exists a command sequence of length ℓ such that we know the position of r afterwards. The above observations yield:

Corollary 9.3.3 *ROBOT LOCALIZATION is NP-hard.*

9.3.2. Merging Two Particles

We start with a special class of polyominoes. We call a polyomino P *simple* if decomposing P with horizontal lines through pixel edges results in a set of rectangles \mathcal{R} such that the edge-contact graph $\mathcal{C}(\mathcal{R})$ of \mathcal{R} is a tree. The edge-contact graph of a set of rectangles in the plane contains a vertex for each rectangle and an edge for each side contact; a corner contact does not result in an edge. A *hole* of a polyomino P is a maximal set of blocked cells (cells not contained in P) that are connected such that there exists a closed walk within P surrounding it. As usual, simplicity of a polyomino captures the feature of not containing holes. A *shortest path* from a pixel p in P to a rectangle R in \mathcal{R} is a shortest path from p to a pixel q in R such that $\text{dist}(p, q)$ is minimal.

Theorem 9.3.4 *For any two particles in a simple polyomino P , there exists a gathering sequence of length D .*

Proof. Let \mathcal{R} be a decomposition of P into rectangles by cutting P with horizontal lines through pixel edges. Then, because P is simple, the edge-contact graph $\mathcal{C}(\mathcal{R})$ of the rectangles \mathcal{R} is a tree. For an example, consider Figure 9.4.

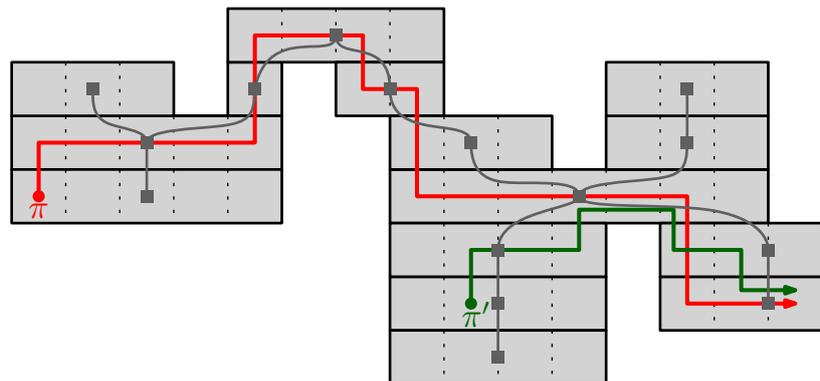


Figure 9.4.: A simple polyomino P , and its edge-contact graph $\mathcal{C}(\mathcal{R})$ (in gray). When the red particle π moves toward the green particle π' , π and π' follow the respective red and green paths. The dotted lines separate the pixels.

For every t , let R_t and R'_t be the rectangles of P containing the two particles π and π' after applying t commands, respectively. Moreover, let S_t be a shortest path from R_t to R'_t in $\mathcal{C}(\mathcal{R})$; and let $S_t(1)$ be the successor of R_t on S_t (if it exists, i.e., $R_t \neq R'_t$).

We use the following strategy:

Phase 1: While $R_t \neq R'_t$, compute a shortest path S_t from R_t to R'_t in $\mathcal{C}(\mathcal{R})$. Move π to $S_t(1)$ via a shortest path in P . Update R_t and R'_t .

Phase 2: If $R_t = R'_t$, merge π and π' by moving π toward π' by a shortest (horizontal) path; note that this gathering sequence merges the particles within R_t .

We now show that this strategy yields a gathering sequence of length D . In fact, the resulting sequence has the following property.

Claim 9.3.5 *For every $s > t$, the rectangles R_s and R'_s are either equal to R_t or lie in the connected component C of $\mathcal{C}(\mathcal{R} \setminus R_t)$ containing R'_t .*

To arrive at a contradiction, assume that π enters a rectangle R_s ($\neq R_t$) that is not in C . Because π moves toward π' in every step, there also exists $p > t$ such that R'_p ($\neq R_t$) is not contained in C .

Because P is simple, there exists an j with $p > j \geq t$ such that $R_j = R'_j = R_t$. However, because $R_j = R'_j$, π and π' merge in R_j , i.e., $R_s = R_t$ and $R_p = R_t$. A contradiction.

The claim implies that π never re-enters a rectangle, i.e., it moves to an unseen rectangle in every step of Phase 1. Because P is finite, π and π' eventually meet in some rectangle which ends Phase 1 and therefore merge in Phase 2.

Moreover, for every t , the merge location and R'_t lie in C or are equal to R_t . Consequently, in every step, π moves toward the merge location on a shortest path. Because a shortest path is at most of length D , the length of the gathering sequence is bounded by D .

This claim implies that the merge location and R'_t lie in C or are equal to R_t . Consequently, in every step, π moves toward the merge location on a shortest path and thus that the gathering sequence is at most of length D . \square

In the remainder, we call the strategy used to prove Theorem 9.3.4 **DYNAMICSHORTESTPATH (DSP)**: Move one particle toward the other along a shortest path; update the shortest path if a shorter one exists. The example in Figure 9.5 shows that DSP may perform significantly worse in non-simple polyominoes.

Proposition 9.3.6 *The strategy DSP may not yield a gathering sequence of length $O(D)$ in non-simple polyominoes.*

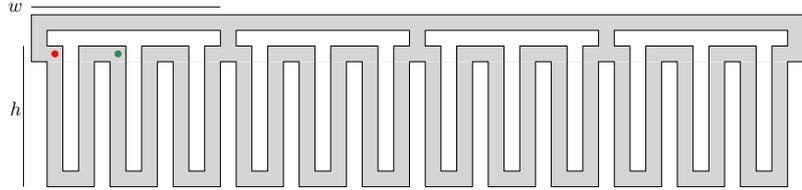
Proof. By the symmetry of P , the distance between the two particles decreases for the first time when one of them is at the left or right side of P . Therefore, denoting the number of holes by H where each hole is of height h and width w as indicated in Figure 9.5, the length of the

gathering sequence C is $H(6h + w) + 3$, while the diameter is bounded by $D \leq (H - 2)w + 6h + 2w + 4 = Hw + 6h + 4$. Choosing $h := cw/6$ for some constant $c \geq H$, the ratio of $|C|$ and $|D|$ can be arbitrarily large:

$$\frac{cHw + Hw + 3}{Hw + cw + 4} \geq \frac{H(c + 1)}{H + c + 1} \geq \frac{H}{2}$$

□

Figure 9.5: When the red particle π moves toward the green particle π' by shortest paths, π visits the entire bottom path.



Nevertheless, DSP always merges two particles;

Proposition 9.3.7 For every polyomino P with n pixels and diameter D and every configuration with two particles, DSP yields a gathering sequence of length $O(nD)$.

Proof. Let π follow π' . We show that within n commands, their distance Δ decreases at least by one. Note that if the shortest path must be updated, the distance decreases.

Consider a sequence C of ℓ steps, in which Δ remains constant. In C , π' has no collision and the shortest path is only updated when π reaches the end of the current shortest path, i.e., a previous position of π' . Let p_0 and p_1 denote the initial position of π and π' , respectively. When π reaches p_i , p_{i+1} denotes the position of π' . Let v be the coordinate vector from p_0 to p_1 . Because π has no collision, v is the coordinate vector from p_i to p_{i+1} for every i . Note that π' must have a collision when moved N times in direction v (because P ends). Moreover, π reaches p_{i+1} from p_i after at most D commands. Consequently, $\ell \leq nD$.

□

Using a different strategy yields a better bound: The strategy `MoveToExtremum` (MTE) iteratively moves an extreme particle (e.g. bottom-leftmost) to an opposite extreme pixel (e.g. top-rightmost) along a shortest path.

Theorem 9.3.8 For any two particles in a polyomino P , MTE yields a gathering sequence of length at most D^2 .

Proof. Let q be the top-rightmost pixel of P . To merge the two particles in q , our strategy is as follows: Identify the particle π that is bottom-leftmost. Apply a command sequence that moves π to q on a shortest path. Repeat.

Claim 9.3.9 In each iteration, the sum of the distances Δ of the two particles to q decreases.

Note that Δ decreases when the other particle π' has a collision. If π' had no collision, there exist a pixel that is higher or more to the right than q , contradicting the choice of q . Consequently, the sum of distances Δ , which is at most $2D$ at start, decreases at least by 1 for every D steps. Hence after $O(D^2)$ steps, Δ is reduced to 0. \square

Note that there exist polyominoes, e.g., a square, where the number of pixels n is in $\Omega(D^2)$. Therefore, Theorem 9.3.8 significantly improves the bound of $O(n^3)$ in [22].

Finally, we note that a shortest gathering sequence for two particles in a non-simple polyomino may need to exceed D .

Proposition 9.3.10 *Let P be a polyomino with two particles. A shortest gathering sequence may be of length $\frac{3}{2}D - O(\sqrt{D})$.*

Proof. Let $h \in \mathbb{N}$. Consider the polyomino P illustrated in Figure 9.6 which consists of the bottom row and S chimneys of height h and length $2h + 4$. Because P consists of $(2h + 4)S + 6S$ pixels, it has a diameter of $D = (h + 5)S$. We set $S = 2h + 4$. The two particles π_1 and π_2 have an initial distance of D such that the number of chimneys to the left of π_1 and to the right of π_2 is $\frac{1}{2}(h - 1)$. Due to the symmetry of P and the

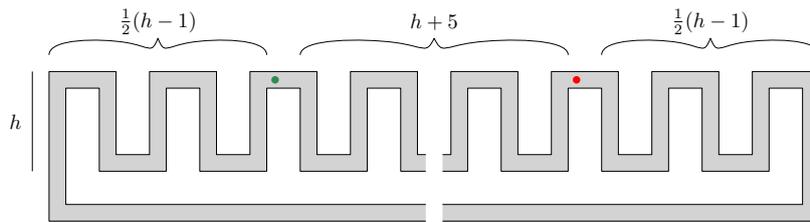


Figure 9.6.: A polyomino consisting of a base and S chimneys.

placement of the particles, the distance Δ of π_1 and π_2 cannot decrease within the first $\frac{1}{2}(h - 1)(2h + 4) - 1 = \frac{D}{2} - O(\sqrt{D})$ commands. Without loss of generality, we assume that π_1 is at the left side of P right before Δ decreases for the first time. We call this configuration *west-touch*.

Starting from the west-touch configuration, the best merge location is in the top of the leftmost chimney. Hence, the distance of π_2 to the merge location is $D - h = D - O(\sqrt{D})$.

Consequently, a gathering sequence is at least of length $\frac{D}{2} + D - O(\sqrt{D})$. \square

9.3.3. Reducing the Number of Particles

Now we show how to significantly decrease the number of particles with few commands to a parameter proportional to the complexity of the polyomino, namely the number of convex corners. This is particularly relevant for establishing the existence of *oblivious* gathering strategies that are capable of merging all particles in an efficient manner, even if their initial configuration is not known. (See Subsection 9.5.1.)

Lemma 9.3.11 *Let P be a polyomino with diameter D and k convex corners. For every configuration $A \in \mathcal{P}$, there exists a command sequence of length $2D$ which transforms A to a configuration $A' \in \mathcal{P}$ such that $|A'| \leq k/4$.*

Proof. We distinguish four types of convex corners; northwest (NW), northeast (NE), southwest (SW), southeast (SE). By the pigeon hole principle, one of the types occurs at most $k/4$ times; without loss of generality, let this be the NW corners.

We show that after applying the sequence $\langle l, u \rangle^D$, every particle lies in a NW corner: Consider a particle π in pixel p . Unless π lies in a NW corner, it moves for at least one command in $\{l, u\}$. Because P is finite, there exists an ℓ large enough such that π ends in a NW corner q when the command sequence $\langle l, u \rangle^\ell$ is applied, i.e., there exists an pq -path consisting of at most ℓ commands of types l and u , respectively. Because a monotone path is a shortest path, it holds that $\ell \leq D$. \square

9.3.4. General Upper Bounds

Combining Lemma 9.3.11 and Theorem 9.3.4 yields:

Corollary 9.3.12 *For a set of particles in a simple polyomino P with diameter D and k convex corners, there exists a gathering sequence of length $O(kD)$.*

Lemma 9.3.11 and Theorem 9.3.8 imply the following fact:

Corollary 9.3.13 *For any set of particles in a polyomino P with diameter D and k convex corners, there exists a gathering sequence of length at most $O(kD^2)$.*

We obtain the analogous result for three-dimensional settings by analyzing cuboids instead of rectangles, six directions of motion instead of four, and the corners in eight quadrant directions instead of four.

9.4. Reinforcement Learning

When looking at the pixilated graphics in Figure 9.11, we are reminded of classical arcade games. In 2013, Mnih et al. [251] proposed an algorithm based on deep reinforcement learning that was able to learn how to play Atari games and eventually reach the level of professional players [252]. Because such algorithms have now become freely available and reasonably easy to use, e.g., *Stable Baselines* [253], we can let them ‘play’ MIN-GATHERING.

In reinforcement learning, an *agent* is exposed to an *environment* in which it has to achieve an objective by performing a sequence of *actions* that manipulate the *state* of the environment. Upon each action, the environment can yield a *reward* (or penalty), or it can terminate. During repeated trials, the agent has to learn a *policy* that maximizes the (expected) total reward by choosing actions based on *observations*, i.e., the perceived states.

Let us denote the state space by \mathcal{S} , the actions by \mathcal{A} , the transition function by $T : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$, and the reward function by $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. For simplicity, this notation neglects probabilistic elements that are common in other use cases. Starting at an initial state $s_0 \in \mathcal{S}$, the agent selects an action $a_0 \in \mathcal{A}$ based on the probabilities of the policy $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, resulting in the reward $r_0 = R(s_0, a_0)$, and leading to the next state $s_1 = T(s_0, a_0)$. This is repeated until a final state is reached, after which the total reward $\sum_i r_i$ is evaluated. We call such a sequence a *period*.

We use a *Convolutional Neural Network* (CNN) on an image of the particle locations as a policy. The construction and training can be performed by readily available algorithms such that we only need to provide the environment, but it is useful to understand the basics of the algorithms before we design the environment. While we could simply display the particles as a matrix and give a reward for successful gathering, the algorithms could not learn efficiently from such an environment. In the following, we give a short introduction into the inner workings of these algorithms.

During training, we need to optimize the (initially random) network parameters to (incrementally) yield better actions. A fundamental problem in reinforcement learning is that we need a sequence of actions to gain a reward, but the policy needs to be trained for individual actions. Often, we even need to perform some penalized actions along the way. How can we compute the advantage of actions not only based on their direct reward or penalty but based on their long-term influence, such that the policy can improve its output? This is known as the *credit assignment problem*, and a common strategy is to consider all future rewards (reduced by some *discount factor*) for each action.

Let us take a look at the simple *REINFORCE algorithms* [254] as an example:

1. Run multiple periods with the neural network (that initially only returns random values) and compute for each returned action the gradients of the network parameters which increase the probability of this action.
2. Compute the advantages of the performed actions. For each period $(s_0, a_0, r_0), (s_1, a_1, r_1), \dots, (s_n, a_n, r_n)$, the advantage of performing a_i in state s_i is determined by $A(s_i, a_i) = \sum_{j=i, \dots, n} \gamma^{j-i} \cdot R(s_j, a_j)$, where $\gamma \in (0, 1)$ is the discount factor. We standardize the advantages by subtracting the mean over all periods and dividing by the standard deviation. Positive standardized advantages now correspond to actions that lead to above average rewards.
3. Multiply the gradients of the first step with the standardized advantages of the second step, and use their mean to perform a *Gradient Ascent* step on the neural network. The more advantageous an action has been, the stronger it gets reinforced.
4. Repeat this procedure until the policy performs sufficiently well.

In our implementation, we use the more advanced *Proximal Policy Optimization* (PPO) algorithm [255], which actually learns not only the best action but also the advantage. The differences between PPO and REINFORCE are significant in practice, but for designing a reasonably good reward function as in Subsection 9.4.1, understanding the idea of the REINFORCE algorithms should suffice. For a deeper understanding,

we refer the curious reader to the extensive current literature, such as [256].

We design the environment for MIN-GATHERING as follows:

- ▶ The observations are images of the particle locations scaled to 84×84 pixels with a maximum filter for all instances. This is a common resolution keeping the CNN reasonably small, which not only speeds up the computations but can also help the CNN to generalize.
- ▶ We fill the environment completely with particles. A motion sequence that gathers all particles can be applied to any configuration. However, it is also possible to use a concrete configuration as the initial state.
- ▶ Each action is repeated automatically a fixed number of times, also called *frame skipping*. This speeds up the learning process drastically, as a few random (repeated) actions can result in visible gathering progress. Otherwise, random actions have only a low chance of making notable progress.
- ▶ We not only provide the four basic motions as actions but also add diagonal motions, which are simulated by two basic motions in random order. This drastically improved the performance in preliminary experiments, especially with much *frame skipping*. Diagonal movements are also a common pattern in the solutions. Providing them directly speeds up the learning process.
- ▶ The lowered resolution and the frame skipping makes it difficult to gather particles to a single location. Thus, we consider the particles as gathered if they are within a radius of 10 steps of an extreme point. The final gathering is then performed by the heuristic MIN-SUMTOEXTREMUM, which only needs around 15 additional motions in our experiments as the particles are already close to an extreme point.
- ▶ If the particles are not gathered after 500 motions for Corridor, 800 motions for Capillary, or 3500 motions for Brain, the period gets terminated. This prevents the algorithm from spending too much time if the agent ‘gets lost’. These limits have been chosen based on the performance of the classical algorithms.
- ▶ The design of the reward function is the most important part, and is discussed in Subsection 9.4.1.
- ▶ The extraction of the best solution directly from the learning process is discussed in Subsection 9.4.2.

9.4.1. Reward

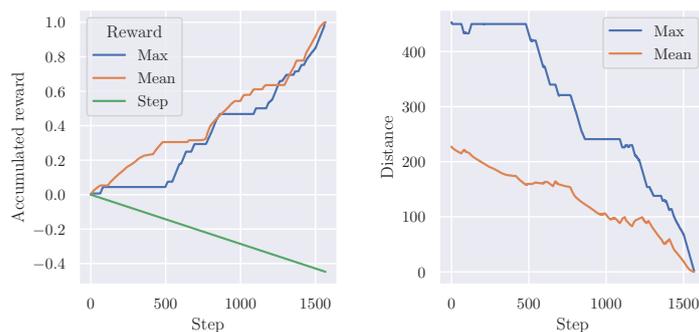
Giving a reward only after all particles are gathered is not practical because every period that does not gather all particles looks equally bad. Using this method, a command sequence that is capable of getting the particles at least close by looks as bad as a command sequence that does nothing at all. This implies that as long as the particles are not gathered by chance, which is very unlikely, all actions are classified as bad. If all action are equally bad, the policy cannot do anything but perform random motions.

A straightforward alternative is to give a reward every time the diameter of the particle swarm is reduced. Of course, we only give a reward if the all-time minimal diameter is reduced; otherwise, repeatedly growing and shrinking the particle swarm may be learned as a ‘good’ strategy. To encourage a short gathering sequence, we can additionally give a small penalty for every action.

An issue with this reward function is that it is time-consuming to compute after every step. We know that extreme points in the maze are good gathering locations. Instead of computing the diameter, we could compute the maximal distance of a particle to an extreme point. Let E be the set of extreme points, then the maximal distance is defined as $\min_{e \in E} \max_{p \in P} \text{dist}(p, e)$. This gives us the option to choose any extreme point, but it must be the same for all particles. The distance of each location to an extreme point is fixed and can be computed ahead of time, thereby making this computation just a lookup for every particle.

For instances like `Brain`, we actually first need to escape some recesses before we can minimize the distance. This has the same issue as before of being hard to achieve through random moves. Here we can support the detection of progress not only by providing reward for minimizing the maximal distance but also by providing reward for minimizing the mean distance.

Because the distances can vary strongly for different instances, we normalize the rewards such that each of the two reward components can only give an accumulated maximal reward of 1, and the motion penalty an accumulated penalty of at most -1 . We normalize the reward of minimizing the maximal or mean distance by 1 by dividing through the initial distance. The rewards are normalized via dividing by the initial maximal resp. mean distance. The motion penalty is normalized by the motion limit L such that each motion gives a penalty of $-1/L$.



(a) Accumulated rewards.

(b) Maximal and mean distance.

Figure 9.7.: Rewards and distances during the best movement sequence of the `Brain` instance, computed with reinforcement learning. We can see that the maximal distance, which we want to minimize, stagnates during the first 500 steps. Maximal and mean distances must both overcome local minima.

An example for the `Brain` instance is given in Figure 9.7. We see that the mean distance supports the maximal distance well as it is more continuous than the maximal distance. The maximal distance only markedly decreases after 500 steps, which is too long without proper feedback. The maximal and mean distances must overcome local minima until the particles are gathered, showing that a naïve local search would not be successful. Even if a local search were to succeed, it is unlikely that the gradients would always point in a direction that leads to a short

gathering sequence. The strength of the reinforcement learning approach, however, is that it automatically improves the reward function.

9.4.2. Implementation

In this section, we show how to easily implement the optimizer. Modern reinforcement learning libraries require barely any knowledge of reinforcement learning or neural networks. Of course, some knowledge and experience is useful, especially for designing a good reward function. This is comparable with modern MIP-solvers that can be used without much understanding of the underlying techniques, but creating a good formulation that can be quickly solved is more difficult. A simple implementation that finds the shortest action sequence to achieve the objective can look as follows:

```

1 # Create a learning environment
2 class LearningEnv(gym.Env):
3     def __init__(self, simulation, limit, repeat):
4         self.simulation = simulation
5         self.rewards = Rewards(simulation, limit)
6         self.repeat = repeat
7         self.min_solution = None
8         # Define input and output for the neural network
9         self.action_space = gym.spaces.Discrete(simulation.no_actions)
10        self.observation_space = gym.space.Box(low=0, high=255,
11            shape = simulation.image_shape, dtype=np.uint8)
12
13    def step(self, action):
14        # Perform a step in the simulation and give feedback.
15        for i in range(self.repeat):
16            self.simulation.step(action)
17            observation = self.simulation.as_image()
18            reward, abort = self.rewards.eval_state()
19            if self.simulation.is_gathered():
20                abort = True
21            new_sol = self.simulation.history
22            if self.min_solution is None or
23                len(self.min_solution) > len(new_sol):
24                self.min_solution = new_sol
25
26        return observation, reward, abort, {}
27
28    def reset(self):
29        # Reset simulation and rewards for the next try.
30        self.simulation.reset()
31        self.rewards.reset()
32
33    # Load instance
34    simulation = Simulation('my_instance.json')
35    # Allow up to 1000 steps and automatically repeat four times
36    env = LearningEnv(simulation, limit=1000, repeat=4)
37    # Automatically resize observations to 84x84
38    resized_env = ResizeObservation(env, shape=(84, 84))
39    # Automatically build neural network (CNN)
40    model = PPO('CnnPolicy', resized_env)
41    # Try for 300 000 steps
42    model.learn(total_timesteps=300_000)

```

```

43 |
44 | # Output best solution
45 | print("Solution:", env.min_solution)

```

This code uses *Stable Baselines 3* [253], but interfaces in state-of-the-art machine learning libraries are highly volatile. In this case, we are using the *Proximal Policy Optimization* (PPO) algorithm, but it can be replaced by multiple other algorithms.

This implementation only misses two problem-specific details: the simulation and the reward function. The simulation needs to be able to perform a sequence of individual actions encoded by discrete numbers (e.g., 0 for up, 1 for right, . . .), return the current states as an image or a matrix, detect if the objective has been achieved, remember the corresponding solution, and reset to the initial state. The reward function only needs to compare the current and the last state and rate the change or abort the current solution process, e.g., by a length limit, if it is not promising.

Note that this approach actually uses the training phase of the neural network to find a solution. Usually, one trains the neural network to use it afterwards. As our environment does not change and the sequences remain valid, simply using the best encountered solution is the better strategy.

Future Work 9.1.

We currently always reset the simulation to its initial state. Thus, the reinforcement learning algorithm will only look at independent complete solutions. If we have some promising intermediate states, it may be more efficient to continue with an intermediate state. This can easily be implemented by loading this state during reset instead of resetting the simulation. The primary challenge is to detect promising states. Can we bring the reinforcement learning approach closer to classical search algorithms like A^* , but with a self-learning distance function using such a technique?

9.5. Evaluation

In this section, we evaluate the performance of the following approaches on practical instances in a simulation.

- ▶ The approach `STATICSHORTESTPATH` (SSP) iteratively merges pairs of particles by moving one to the position of the other, along a shortest path, see Alg. 2 in [22].
- ▶ The approach `DYNAMICSHORTESTPATH` (DSP), as described in Section 9.3.
- ▶ The approach `MOVETOEXTREMUM` (MTE), as described in Section 9.3. Among the four possible commands, we choose an extremum that minimizes the initial sum of distances to both particles.
- ▶ The heuristic `MINSUMTOEXTREMUM` (MSTE) generalizes the idea of MTE. It selects an extremum with the smallest initial sum of distances to all particles and iteratively performs a command that decreases this sum the most. If no command decreases the sum,

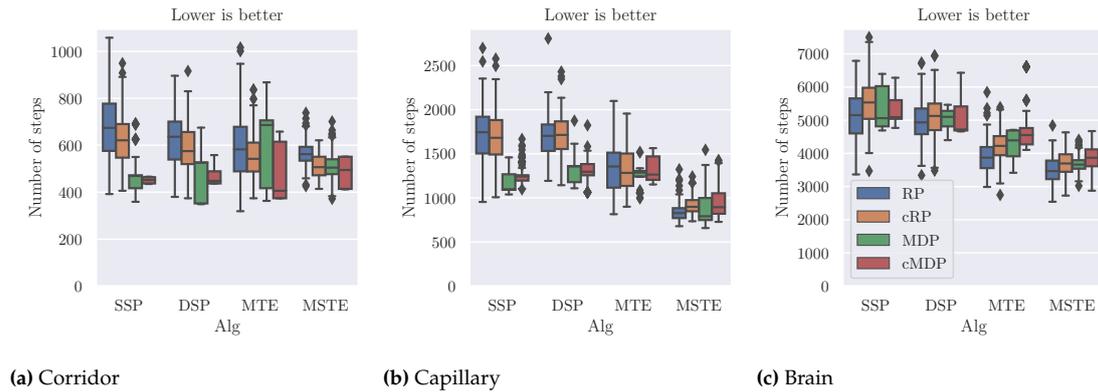


Figure 9.8.: Comparison of the combinatorial algorithms with different pair selections (random (RP) or most distanced (MP)) and optional corner preprocessing (cRP resp. cMP).

two particles are selected and merged by MTE. Afterwards, MSTE resumes.

- Additionally, we evaluate the machine learning approach REINFORCEMENTLEARNING (RL), as described in Section 9.4. We use the default parameters for all mazes and only vary the motion repetitions, limits, and time steps. The Corridor and the Capillary maze are trained over 300 000 time steps with a frame skipping of 4. The Brain maze is trained over 600 000 time steps with a frame skipping of 16.

The experiments were performed on over 130 random particle configurations with 1000 particles in each environment. Every configuration was solved by all strategies to ensure comparability. The preprocessing used a random direction to move the particles into corners. We used a workstation equipped with an AMD Ryzen 7 1700 CPU with 8×3.0 GHz and 32 GB memory, and an Nvidia GTX 1050 Ti GPU with 4 GB memory.

We first compare the combinatorial strategies SSP, DSP, MTE, MSTE and their options. For these strategies, we evaluate the options of

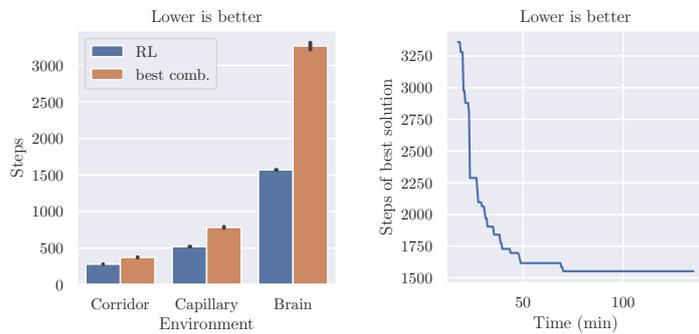
1. choosing a pair uniformly at random or
2. choosing the pair with maximal distance.

Additionally, we analyze the advantage of using the preprocessing strategy that moves all particles to corners, as described in Subsection 9.3.3.

The results in Figure 9.8 show that MSTE performs best on average, and that the pair selection and preprocessing options only have a small influence on it. Only for the small Corridor instance, the SSP strategy with most distanced pairs and preprocessing performs visibly better. For the shortest path strategies SSP and DSP, the most distanced pairs show a significant advantage for the first two environments, while it has only a small influence on MTE and MSTE.

We also experimented with optimal pair merging sequences computed by an A^* -algorithm, but in preliminary experiments we encountered worse results at a higher computational complexity such that we ignored this approach for the final experiments. An explanation for the worse results could be that, e.g., MTE and MSTE are guiding many more particles than just the pair to the extreme position and, thus, are more efficient for

gathering all particles even if the strategy may be suboptimal for just the pair.



(a) RL vs. best combinatorial algorithm. (b) Improvement of RL for Brain over time.

Figure 9.9.: The reinforcement learning approach compared to the best results of any of the combinatorial algorithms. The reinforcement learning approach shows to be superior especially for the complex environments. It already yields superior solutions after a few minutes.

When comparing the best solution of any of the previous algorithms with the solution returned by the reinforcement learning approach in Figure 9.9a, the reinforcement learning approach shows clear superiority despite assuming a fully-filled environment. Especially for the Brain environment, the reinforcement learning approach yields command sequences of less than half the length of any of the other algorithms. The reinforcement learning approach needs over two hours to train, compared with just a few minutes for the execution of MSTE, but already after a few minutes it yields superior solution that improves further over time as can be seen in Figure 9.9b. Additionally, the runtime can be improved further by using parallel agents, as PPO supports parallel optimization.

Overall, the reinforcement learning approach is superior and can deal much better with the complex particle configurations than our combinatorial algorithms. Contrary to the combinatorial algorithms, it can also easily utilize parallelization. In his master's thesis, Konitzny [257] performs a deeper and more extensive analysis of this technique and achieves even better results by lower level optimizations, e.g., replacing the activation neurons. Contrary to our work, he actually uses the trained agent to perform the gathering; this also allows non-deterministic environments with continuous physics, showing the flexibility of this approach.

Future Work 9.2.

Can we achieve a similar performance to deep reinforcement learning using a more extensive local search? We tried approaches that looked multiple steps into the future, but they barely improved the overall performance. The primary problem seems to be rating the individual states. Maximal and average distances do not seem to work well enough (based on some preliminary experiments). The advantage of the deep reinforcement learning approach is that it learns a more advanced rating automatically.

9.5.1. Oblivious Merging

In practice, it may be costly to determine the position of the individual particles; therefore, *oblivious* approaches that do not need this information can be of interest. Such a setting is equivalent to the situation where initially, each pixel contains a particle; a gathering sequence for all particles is certainly a gathering sequence for any other (partial) initial distribution of particles. Recall Corollary 9.3.3, implying that this problem remains NP-hard. In order to estimate the cost of this restriction in practice, we study how the number of populated grid cells behaves over time, depending on the initial number of particles; see Figures 9.10 to 9.12.

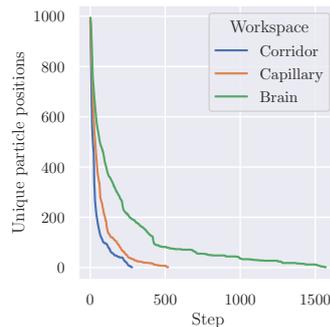


Figure 9.10.: Number of particle groups over time using RL for 100 random configurations each of 1000 particles. For all workspaces, the number of unique particle positions drops very quickly, and collecting the last 100 particles comprises the majority of the steps.

Because the number of populated grid cells decreases very sharply in the beginning and almost all steps are used to merge the few remaining groups of particles, we can conclude that missing knowledge of the position of the individual particles has negligible cost for uniform distributions.

9.6. Conclusions

We have described a spectrum of methodological progress on an important problem of great practical relevance. This exposition focuses on two-dimensional scenarios, but a generalization to three-dimensional settings appears to be straightforward. In addition, we point out three other relevant directions for future research.

Firstly, our algorithmic simulations indicate the strength of our methods. However, the different outcomes for deterministic as well as ML approaches indicate that further, more detailed algorithmic studies are warranted to understand the most successful line of attack; this includes studies of the necessary tradeoff between computation time and number of actuation steps, but also includes modified models in which an actuation step may be able to move particles by more than an elementary distance. Secondly, how can we deal with random errors in actuation and navigation? Our insights into oblivious methods clearly indicate that these should remain tractable, but more detailed considerations for frequency and amount of errors should provide quantifications and error-correcting approaches. Finally, it is typically not necessary for our application scenarios to gather *all* particles in a target area; moving an appropriate fraction should usually suffice. Figure 9.10 visualizes a slightly different aspect, but still highlights the prospect that a considerably reduced number of actuation steps may be achieved.

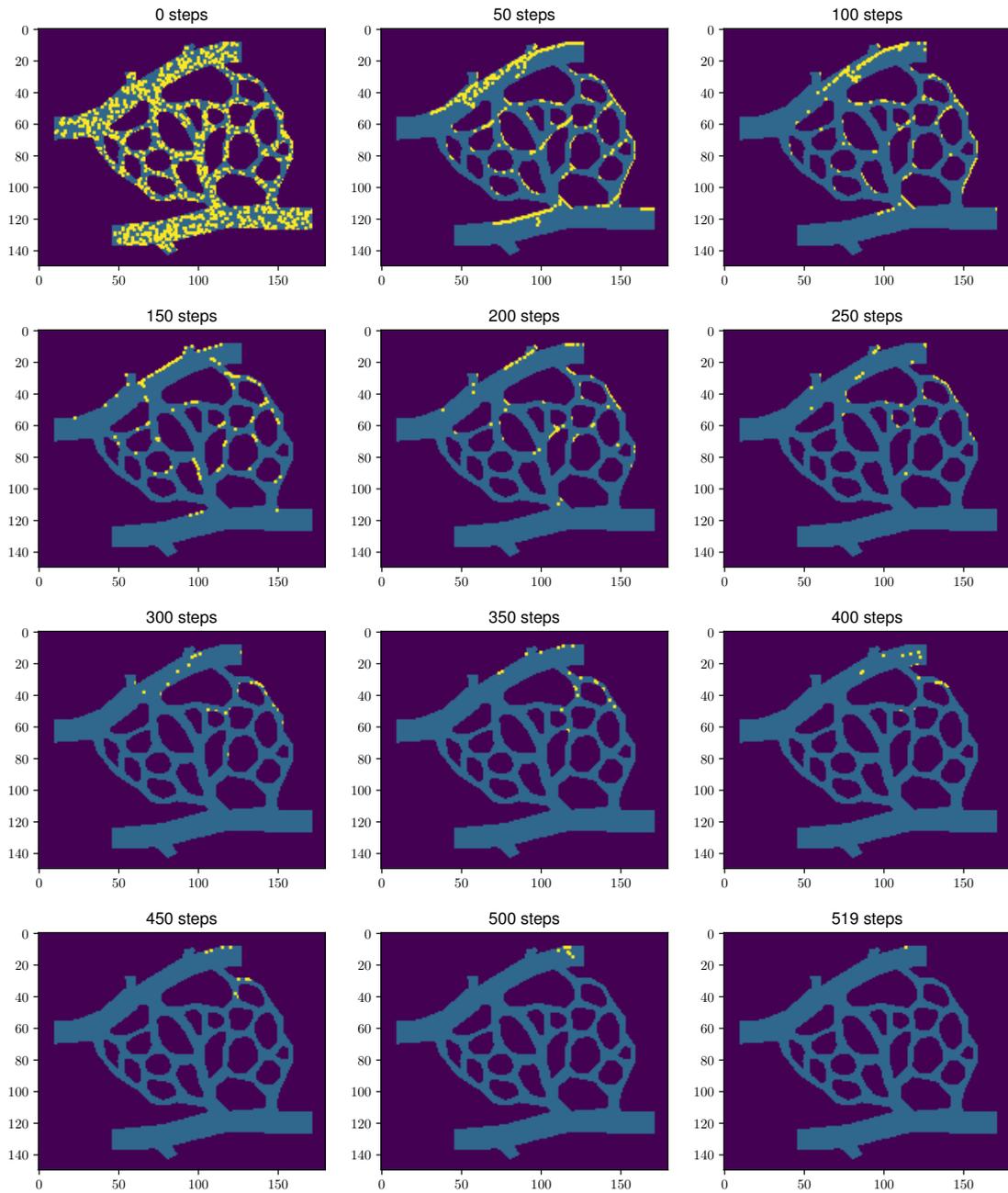


Figure 9.11.: The process of gathering 1000 particles in `Vessel` with RL. The visualization uses a maximum filter to improve visibility. Gathered particles aggregate to a single particle. We see that the particles quickly collapse to a few clusters.

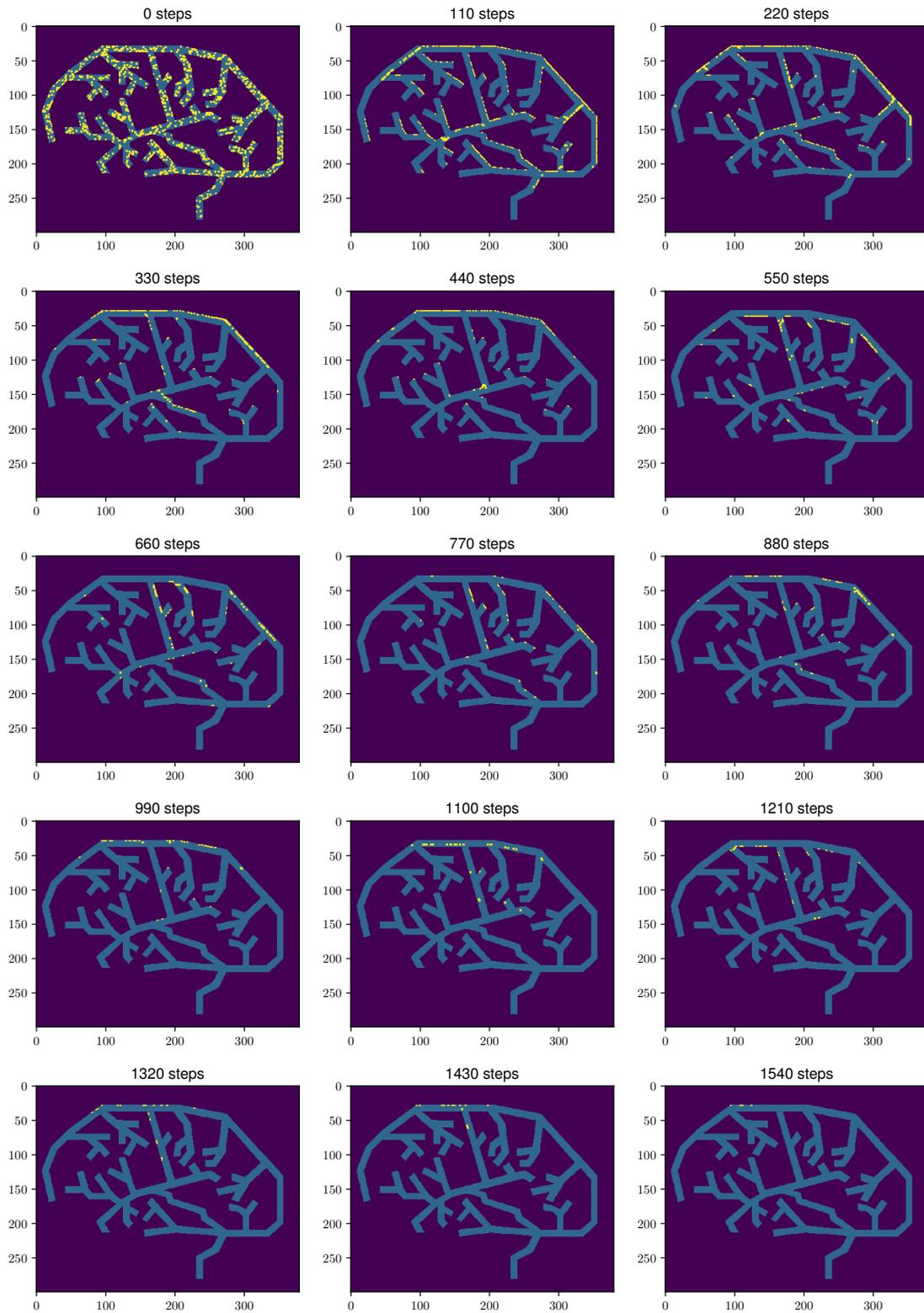


Figure 9.12.: The process of gathering 1000 particles in Brain with RL. The visualization uses a maximum filter to improve visibility. We see how the algorithm quickly moves the particles out of the branches, but sometimes a particle drops back in.

Part V.

CG:SHOP CHALLENGES

In this last part of the thesis, we take a completely different approach to algorithm engineering: instead of solving the problems ourselves, we let others solve them in a competition setting. Competitions have become a popular element in computer science. Examples are Google's yearly coding competition, *hash code*, which reported over 125 000 registered participants in 2021; or Kaggle (www.kaggle.com) which has run hundreds of competitions in machine learning, with a huge community of more than a million registered users. Specialized communities offer their own competitions such as the *DIMACS Implementation Challenges* (<http://dimacs.rutgers.edu/programs/challenge/>), the *Graph Drawing Contest* of the *International Symposium on Graph Drawing and Network Visualization*, or the SAT-competitions (<http://www.satcompetition.org/>). Apart from competitions, there are also static benchmarks for some problems such as the TSPLIB [198] for the TRAVELING SALESMAN PROBLEM, or the MIPLIB [258] for mixed integer programming.

The Computational Geometry community has been lacking such competitions, despite having an avid practical community. The CG:SHOP Challenges have been created to fill this gap and have since become part of the *CG Week* that also hosts the Symposium of Computational Geometry (SoCG), considered the flagship conference in Computational Geometry.

In Chapter 10 (On Hosting the CG:SHOP Challenges) on page 227, we provide detail on the development and hosting of the CG:SHOP Challenges. Afterward, we look into the individual competitions. The first challenge is considered in Chapter 11 (CG:SHOP Challenge 2019) on page 243, the second in Chapter 12 (CG:SHOP Challenge 2020) on page 255, and the third in Chapter 13 (CG:SHOP Challenge 2021) on page 261.

On Hosting the CG:SHOP Challenges

10.

This chapter gives insight into the development and hosting of the CG:SHOP Challenges. We look at the architecture of the supporting infrastructure, the involved tasks and workloads, how to select a good set of instances, and the lessons learned during the first three installments of the competition.

10.1. Introduction

Hosting a competition is a significant amount of work, but under the right conditions it can be a worthwhile enrichment for the community. First, it encourages algorithm engineering research for a problem and may lead to scientific advances. A competition provides not only motivation and a predictable time frame but also recognition. Second, competing in a group is a great team-building exercise that indirectly fosters further scientific progress. Third, it can introduce students into the community.

The last CG:SHOP Challenges have attracted many student groups, and students have become a main focus of ours. A timespan of several months allows the integration of the competition into the semester and enables teachers to offer the participation as a lab class. Additionally, we introduced a junior category so that students are not intimidated by competing against more senior teams. The problems are also designed to be beginner friendly and not require advanced knowledge in Computational Geometry to get started.

The CG:SHOP Challenges were initiated as a workshop at CG Week 2019. The first competition, CG:SHOP 2019, opened February 28th, 2019 and closed May 31st, 2019. It considered the problem of optimizing the area of a polygon on a fixed set of points. Details of the competition as well as its outcome are discussed in Chapter 11 (CG:SHOP Challenge 2019) on page 243. The competition has been well received and afterward, the competition became an official part of *CG Week*. The three top teams of each competition are invited to submit and present an abstract about their approach that is included in the proceedings. For the second competition, CG:SHOP 2020, the timespan was extended to run from September 30th, 2019 to February 14th, 2020. It considered the problem of partitioning a point set into convex areas, which is discussed in detail in Chapter 12 (CG:SHOP Challenge 2020) on page 255. The third challenge, CG:SHOP 2021, considered parallel motion planning, and is discussed in Chapter 13 (CG:SHOP Challenge 2021) on page 261. The fourth challenge, CG:SHOP 2022, is currently running at the time of writing and considers intersection-free partitioning of graphs.

In the following, we first describe the server architecture and development, including the used frameworks and components. This can be valuable information if you plan to develop your own competition page from the ground up. The techniques are state-of-the-art in 2021 at the time of writing, but in a few years there may be other more advanced frameworks.

10.1	Introduction	227
10.2	Server Architecture . . .	228
10.2.1	Frameworks and Li- braries	228
10.2.2	Components	231
10.3	Workload and Tasks . .	234
10.3.1	Problem Selection	234
10.3.2	Development	235
10.3.3	Instance Selection	236
10.3.4	Support	236
10.3.5	Maintenance	236
10.3.6	Analysis	237
10.4	Instance Selection . . .	237
10.5	Lessons Learned	241

Afterward, we discuss the workload involved with hosting a competition. If you are planning to host a competition yourself, you can use this to estimate the required time and energy. Then we look into how one can select a good set of instances using techniques from data science. Finally, we close this chapter with lessons learned.

10.2. Server Architecture

The competitions are managed over a dedicated webpage and server architecture. The start page gives an overview over announced, running, and past competitions as well as recent news, see Figure 10.1. When clicking on a competition, competition-specific information and actions are shown as in Figure 10.2. The next two sections discuss which frameworks and libraries have been utilized, and which components had to be developed.

10.2.1. Frameworks and Libraries

The webpage is powered by a set of different frameworks and libraries on different servers. The usage of such software not only simplifies the development but also prevents security and reliability issues. The general architecture can be seen in Figure 10.3. There are three types of servers: the web server that hosts the webpage, the watcher that looks for failures of the web server, and the verification workers that verify submissions and perform other heavy tasks. In the following, we describe the most important software used, and why we decided to use them.

Python: We selected Python as primary language, as Python provides all necessary tools and is very common among students, easing the training of student assistants. Another advantage of Python is the reasonably easy integration of native C/C++-code.

Django: There are multiple web-frameworks for Python. We chose Django because it provides many important functionalities like database management and migration, user management, security features, and a powerful template language. It is reasonably easy to use and provides a lot of tutorials as well as an avid community. The important basics can usually be mastered within a few days by a student assistant with basic Python experience.

Ubuntu Linux LTS: We use an Ubuntu Linux with long term support as server operating system. The only important decision here is to have long term support because we want a stable software and few changes. There is no need for up-to-date software as long as it has no security issues.

Docker: To make the interaction with the database, message broker, etc., more stable and locally testable, we use Docker containers. The most important aspect of this is the ability to exactly replicate the server for debugging and testing. This also encapsulates the compatibility of critical components, especially the database, from the operating system updates. Database incompatibilities, either due to a different testing environment or due to an update, can be painful to diagnose and fix.

CG:SHOP Help Competitions krupke

Computational Geometry: Solving Hard Optimization Problems Geometric Optimization Challenges

We host optimization challenges for computational geometry problems. The challenges are part of CG Week.

News view all

08/26/2021 1:20 p.m. Unfortunately, there is a slight imprecision in the description of the solution format. If an instance contains three points v, w, x that lie on a line in that order, the overlapping edges vw and vx are considered intersecting. We will update the description shortly.

08/19/2021 10:19 a.m. CG:SHOP 2022 Example Instances released!

07/21/2021 12:04 p.m. Dear colleagues,
We are happy to announce the Fourth Geometric Optimization Challenge, as part of CG Week in Berlin, Germany, June 7-10, 2022.
See <https://cgshop.libr.cs.tu-bs.de/competition/cg-shop-2022/#problem-description>

Announced Competitions

CG:SHOP 2022 Starts on: Sept. 19, 2021, midnight UTC
Organized by: **Sándor Fekete, Phillip Keldenich, Dominik Krupke, Stefan Schirra**
The Fourth Geometric Optimization Challenge is part of CG Week in Berlin, Germany, June 6-10, 2022. The task is to solve the Minimum Partition into Plane Subgraphs Problem.
Review the [problem description](#) for more details.

Past Competitions

CG:SHOP 2019 Ended on: May 31, 2019, midnight UTC
Organized by: **Erik Demaine, Sándor Fekete, Joseph S. B. Mitchell**
First CG:SHOP competition as part of a workshop at CG Week 2019. The problem was to compute polygons with minimal or maximal area for a given point set.
Review the [problem description](#) for more details.

CG:SHOP 2020 Ended on: Feb. 14, 2020, 11:59 p.m. UTC
Organized by: **Erik Demaine, Sándor Fekete, Phillip Keldenich, Dominik Krupke, Joseph S. B. Mitchell**
The Second Geometric Optimization Challenge is part of CG Week in Zurich, Switzerland, June 22-26, 2020. The task is to solve the Minimum Convex Partition Problem, which asks for a set of edges connecting a given set of points which partitions the convex hull of the points into the minimum number of convex regions.
Review the [problem description](#) for more details.

Figure 10.1.: The start page of the web-page gives an overview of the latest news and the competitions.

PostgreSQL: We use PostgreSQL as a relational database. Nearly all database interaction is managed by Django after the basic setup. PostgreSQL is installed as a Docker container. Any well-supported database, like MariaDB, would also be acceptable.

Nginx: Django comes with its own web server, but it is only intended for development. For the use on the production level, a proper web server like Nginx is needed. Nginx provides not only significant speedups but also HTTPS. Apache is an alternative option, but the documentation of using Nginx with Django is more detailed.

CGAL: The verification of the submissions must be exact. Unfortunately, many geometric problems tend to have numeric issues. The exact arithmetic of CGAL prevents such problems while still performing reasonably fast. The disadvantage is that the verification tool has to be written in C++, but Python-alternatives are too slow. Also without CGAL, the usage of a fast programming language like C++, C, or Rust is recommended if the instances are large.

Celery: Celery is a simple and reliable Python framework for distributing tasks among multiple computers. We use it to distribute the verifications of the submissions to the verification workers. Celery

CG:SHOP 2021

Organized by: Sándor Fekete (TU Braunschweig), Phillip Keldenich (TU Braunschweig), Dominik Krupke (TU Braunschweig), Joseph S. B. Mitchell (Stony Brook University)

30 teams participating
Nov. 20, 2020, midnight (UTC) - Feb. 15, 2021, 11:59 p.m. (AoE)

This year's winners will present their approaches in the Challenge session of CG Week at 12:40-13:40 EST Wednesday (June 9). 05/18/2021 6:35 a.m.

The competition has ended. To view your score and the score of the best teams, please refer to the ranking tab.

Announcement Rules Instance Format Ranking Your Teams Your Progress

Coordinated Motion Planning

We are happy to announce the Third Computational Geometry Challenge, as part of CG Week in Buffalo, USA, June 7-11, 2021.

As in previous years, the objective will be to compute good solutions to instances of a difficult geometric optimization problem. The specific problem chosen for the 2021 Challenge is *Coordinated Motion Planning*, as follows.

Description

Given a set of n axis-aligned unit-square robots in the plane, a set $S = \{s_1, \dots, s_n\}$ of n distinct start pixels (unit squares) of the integer grid, and a set $T = \{t_1, \dots, t_n\}$ of n distinct target pixels of the integer grid. During each unit of time, each robot can move (at unit speed) in a direction (north, south, east or west) to an adjacent pixel, provided the robot remains disjoint from all other robots during the motions.

This condition has to be satisfied at all times, not just when robots are at pixel positions. For example, if there are robots at each of the two adjacent pixels (x, y) and $(x + 1, y)$, then the robot at (x, y) can move east into position $(x + 1, y)$ only if the robot at $(x + 1, y)$ moves east at the same time, so that the two robots remain in contact, during the movement, but never overlap.

In addition, for some instances, there may be given a set of obstacles, consisting of a number of stationary, blocked pixels that cannot be used by robots at any time.

The task is to compute a set of feasible trajectories for all n robots, with the trajectory for robot i moving it from s_i to t_i .

Start positions are shown in light green, target positions in solid red.

Objectives

Figure 10.2.: The competition page provides a clean interface to access all important information and submit solutions.

is very dynamic and allows the easy addition and removal of workers. Workers use the same code as the web server but are initiated differently. Making a task executable by a worker is as simple as adding a decorator to the corresponding function. Celery does not distribute files, so the submissions to be verified must be transferred via a separate secure channel. The database can be accessed directly even using Django's middleware.

RabbitMQ: RabbitMQ is a message broker, installed as a Docker container, for managing the communication of Celery with the workers. It was chosen because it is claimed to be more robust than Redis. Redis is claimed to scale better, but we will not receive that many submissions that this would become an important factor.

Nagios: Nagios watches the web server and informs us automatically via email about downtimes or other problems, e.g., expiring SSL-certificates. It runs on a separate server so that it is not influenced by a complete outage of the primary server. Nagios frees the administrators of frequent manual checks and keeps downtimes low. In addition to Nagios, Django is also configured to mail a stack trace whenever an exception is thrown.

Bootstrap: Bootstrap is a great CSS framework that allows the easy construction of attractive web interfaces. While it does not give the webpage a unique design, it makes it look clean and modern without much effort.

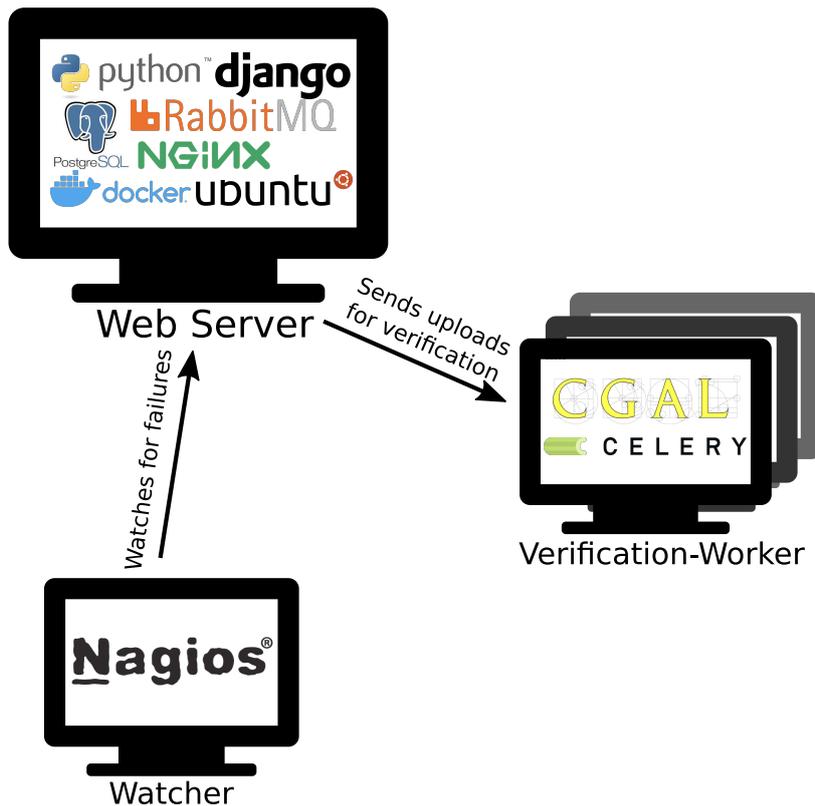


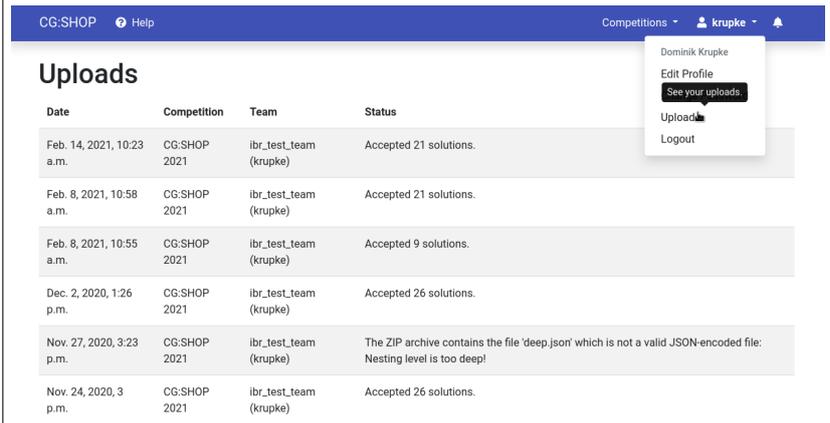
Figure 10.3.: The server architecture consists of a web server, a watcher, and a dynamic set of verification workers. The logos are intellectual property of the corresponding projects.

10.2.2. Components

Using the described frameworks and libraries, a set of components had to be designed and implemented. The following list provides a short description of these.

User Management: Users are more than just an email address and a password. Our users for example usually have an affiliation. Because this event only happens once a year, passwords easily get forgotten, making options for password resets necessary. Extending the basic user system of Django by such additional features is reasonably simple by following corresponding tutorials, as similar features are also required by many other projects.

Group Management: Users participate as a group in a competition. While individual participation in a one-man-group is possible, most participants work in groups, and we want to encourage this by designing the user interface correspondingly. The users need an option to create groups and add members. To keep things simple, we allow every group member to perform any changes and actions for a group, including removing other members. This may allow malicious acts by a single group member, but these can be undone by an administrator. Because of high fluctuation in the academic world, groups are associated with a single competition and frozen some time after the deadline. This allows us to freeze the association and student-status of each group at the time of participation. We also included options to let a team remain anonymous until they want to become visible. We had to implement the group management by ourselves.



Date	Competition	Team	Status
Feb. 14, 2021, 10:23 a.m.	CG:SHOP 2021	ibr_test_team (krupke)	Accepted 21 solutions.
Feb. 8, 2021, 10:58 a.m.	CG:SHOP 2021	ibr_test_team (krupke)	Accepted 21 solutions.
Feb. 8, 2021, 10:55 a.m.	CG:SHOP 2021	ibr_test_team (krupke)	Accepted 9 solutions.
Dec. 2, 2020, 1:26 p.m.	CG:SHOP 2021	ibr_test_team (krupke)	Accepted 26 solutions.
Nov. 27, 2020, 3:23 p.m.	CG:SHOP 2021	ibr_test_team (krupke)	The ZIP archive contains the file 'deep.json' which is not a valid JSON-encoded file: Nesting level is too deep!
Nov. 24, 2020, 3 p.m.	CG:SHOP 2021	ibr_test_team (krupke)	Accepted 26 solutions.

Figure 10.4.: Under 'Uploads', the users can check the status of all their uploads as well as those from team members. In case of problems, the status message provides an explanation.

Competition Management: Competitions need a management system to organize timing and information. Competitions can take on different forms, e.g., offering different tracks, which needs to be reflected by the designed system. The challenge is to make the components generic enough such that we do not need to redevelop too much for every new competition, while still allowing for specializations. This has been primarily accomplished by making the components easily copyable. A new competition is created by cloning a previous one and simply exchanging the key, the content and information, and the verifier. Every generic logic uses the competition key so that the data does not get mixed up. The template and app system of Django provides an excellent structure, so that finding and replacing the necessary parts is reasonably simple. For example, the competition description is a single file that only contains the description and gets embedded into the competition page, which, in turn, gets embedded into the main page.

Submission System: The submission system is generic and can easily be reused by any competition, by simply replacing a key in a code fragment and adding a verifier for the competition. Every submission has to be a single zip-file which should be generic enough for nearly every competition. The content of this zip has to be verified by the competition-specific verifier which is part of the verification system. Whenever a zip has been uploaded, the verification system is informed. A user can get an overview of their status in their account as shown in Figure 10.4.

Notification System: Communication is important. This provides transparency and helps the user to get confidence in working with our system quickly. The messages give them direct feedback as to whether or not they did everything correctly. At the same time, we do not want to spam the user. CG:SHOP informs users with a subtle red badge about new notifications, see Figure 10.5. Once clicked, the badge is removed and the message list pops up, as in Figure 10.6, allowing further clicks to read the full notifications.

News System: The important announcements are made via community mailing lists to reach a large audience. For smaller announcements, we use a blog-like news system. An important motivation for this was to create a possibility to notify discretely about corrections and server maintenance.



Figure 10.5.: A red badge subtly informs about new notifications without annoying the user.

The screenshot shows the CG:SHOP website interface. The main content area features a header with the site name and navigation options. Below the header, there is a section titled 'Computational Geometry: Solving Hard Optimization Problems' with a sub-section 'Geometric Optimization Challenges'. A news section follows, listing several updates with dates and times, each accompanied by a 'read more' link. At the bottom, there is an 'Announced Competitions' section for 'CG:SHOP 2022', listing organizers and the event details.

The notification panel on the right side is titled 'Notifications' and contains a list of messages:

- Upload accepted.** 14.02.21, 10:30 a.m. krupke uploaded (2021-02-14 10:23 UTC) a file 'submission.zip' with 21 solutions which have success...
- Upload successful!** 14.02.21, 10:23 a.m. Your submission will be verified soon. You can check the status under <https://cgshop.libr.cs.tu-bs.d...>
- Upload accepted.** 08.02.21, 10:58 a.m. krupke uploaded (2021-02-08 10:58 UTC) a file 'submission.zip' with 21 solutions which have success...
- Upload successful!** 08.02.21, 10:58 a.m. Your submission will be verified soon. You can check the status under <https://cgshop.libr.cs.tu-bs.d...>
- Upload accepted.** 08.02.21, 10:56 a.m. krupke uploaded (2021-02-08 10:55 UTC) a file 'submission3.zip' with 9 solutions which have success...
- Upload successful!** 08.02.21, 10:55 a.m. Your submission will be verified soon. You can check the status under <https://cgshop.libr.cs.tu-bs.d...>
- Upload successful!** 02.12.20, 01:26 p.m. Your submission will be verified soon. You can check the status under <https://cgshop.libr.cs.tu-bs.d...>
- Problems with your submission.** 27.11.20, 03:23 p.m. Your submission Upload of deep.zip by krupke at 2020-11-27 15:23:02.061432+00:00 (UTC) has been rej...
- Upload successful!** 27.11.20, 03:23 p.m. Your submission will be verified soon. You can check the status under <https://cgshop.libr.cs.tu-bs.d...>
- Upload successful!** 24.11.20, 03:00 p.m. Your submission will be verified soon. You can check the status under <https://cgshop.libr.cs.tu-bs.d...>

At the bottom of the notification panel, there is a link to 'View all notifications'.

Figure 10.6.: Detailed notifications on the webpage. Mails would have been simpler to implement but require more time and care due to the frequency.

Admin System: Django comes with a powerful administration interface that allows easy alterations to the database. Additionally, Django can also give us direct access to the data, so that the progress of the participants can be easily analyzed. We are currently developing the option to automatically create daily reports based on Jupyter-Notebooks. An administrator can create one or more Jupyter-Notebooks computing plots and statistics that get automatically reevaluated every day with fresh data. When the plots show abnormalities, the administrator can still download the data and the Jupyter-Notebook and perform further analysis locally. Due to security reasons, we do not allow editing of the Jupyter-Notebooks dynamically on the server. Giving an administrator these automatic insights reduces a lot of stress induced by the unpredictability of such a competition.

Solution Management: Solutions and the corresponding scores need to be saved in a generic and simple way. Some users upload many solutions very frequently, and we have to keep track on the best solutions. It is also interesting to know how the solutions improve over time. Due to the mass of solutions, they must be actively managed; but we cannot simply go through all solutions every time we want to compute the scores. Additionally, some submissions may be rejected due to a mistake on our side, and reevaluating all solutions can take multiple hours or even days. Thus, we need to be able to simply add and remove submissions and their solutions from the system. The primary challenge of the solution management is to make the database fast enough, while memorizing the temporal differences.

Verification System: Every competition needs its own verification tools. The verification consumes a lot of resources and should not be

performed on the web server itself. Geometric problems often tend to have arithmetic problems requiring exact arithmetics, as provided by CGAL. This exact arithmetic comes with the disadvantage of unpredictable memory usage that can lead to out-of-memory errors. The outage of a single verification worker should, hence, not harm the overall system. While we can often detect and fix numerical problems in practice using heuristics, we cannot rely on heuristics for evaluation. The primary risk would be missing some edge cases and rejecting superior solutions, destroying the faith in a fair and competent evaluation. The bulk of the uploads happens right before the deadline, and participants anticipate the results. Therefore, it is useful to be able to scale the resources up: the usage of celery allows us to dynamically add and remove verification workers even on hard reboots.

10.3. Workload and Tasks

In this section, we take a look at the tasks and workloads that were necessary for hosting the CG:SHOP competition for three years (and counting).

10.3.1. Problem Selection

Before we can start a competition, we need a suitable optimization problem as challenge. This must be sufficiently difficult to solve but should also not require too much expertise. Optimally, the problem has a convincing motivation but has not yet gathered too much attention. For example, the TRAVELING SALESMAN PROBLEM would be a terrible problem, because it is unlikely that anyone would be able to beat Concorde [134]. Luckily, a call for problems and an experienced advisory board can perform this task with reasonably low effort (the biggest problem being the synchronization with all members). It is recommended to interleave the problem selection with a preliminary instance selection, which can take more time and is discussed later.

Score

Every problem definition also needs a score by which the participants are compared. Using only the objective function, we can compare solutions for individual instances but not a whole instance set. For the first two competitions, we used a lower resp. upper bound to scale the objective values between 0 and 1. A score can then be computed by simply summing up these values. However, if the bound is not close to the optimal value, the score can get skewed, as can be seen for the MIN-AREA objective in CG:SHOP 2019. Another problem arises if the objective values are relatively tight, leading to very close scores as in CG:SHOP 2020. For CG:SHOP 2021 we used a relative score that uses the best solution as bound, because we had no good bound available. The problem with such a score is that the score for a team can decrease when other teams submit better solutions. While we showed the score progress to the participants in the first two years, we only showed the progress on the objectives of

individual instances in the third year. Much time has been dedicated to designing fitting scores, without a completely satisfying result. Aspects as how should a team with few significantly better solutions be rated compared to a team with many slightly better solutions are difficult to balance in advance.

10.3.2. Development

The development requirements can be split into (1) the generic webpage and (2) the problem-specific verifier. Implementing the webpage is largely a one-off investment while implementing the verifiers is a recurrent task.

Webpage

Providing just a simple web interface to upload files can be done quickly by just following tutorials. Even adding some simple user management and showing plots and tables regarding one's own submissions is straightforward in Django. The verification can be done by a separate process that scans for new submissions, verifies them, and writes the results into the database. This allowed us to implement the webpage of CG:SHOP 2019 within two man-weeks despite having nearly no prior experience in web development.

However, this simple webpage only provided the most essential features and, e.g., did not allow us to add further competitions at a later point in time. For the second competition, we decided to redevelop the webpage from scratch to improve the user experience, and to make the webpage reusable for future years. The verification process was especially problematic and was replaced by the dynamic and distributed system we have now. Thanks to the experience from the first year as well as having more preparation time, we were able to come up with a concrete concept that allowed us to describe and distribute smaller work packages to student assistants. The more complex architecture and many custom elements made the development process significantly more time-consuming. Overall, it approximately cost us three man-months of a researcher, plus a total of six man-months of student assistants.

Verifier

The verifier has to be redeveloped anew for every competition, but as verifying solutions is an essential task of algorithm engineering, it is a task we were familiar with. Implementing a single verifier probably took no longer than one or two days at most. Making them publicly available as Python-modules on PyPI was more time-consuming. This required not only more extensive documentation but also packing the native parts for various platforms. A further issue was that special care had to be put into providing good error messages. There had been multiple special cases, especially in clearly not optimal solutions, that had not occurred to us. While these cases were usually still caught, the error messages were simply unsuitable. Addressing these issues cost us approximately one to

two weeks per verifier. Keeping the verifier private could save time in the future.

10.3.3. Instance Selection

The selection of the instances for the benchmark had been very simple for the first two competitions, as the instances were point sets and we could simply mix some preexisting sets. In the second competition, we noticed halfway through that collinear points can have a significant influence on the objectives. The scores of the teams at this point in time were very close by, and the instances barely contained collinear points. To make the competition more interesting, we generated an additional instance set with highly collinear instances. While it added some action, adapting the competition was not well received by all teams and probably did not change the outcome, as we will see later.

For the third competition on parallel motion planning, new generators had to be devised from scratch. The problem was not only to generate random instances, but instances that included interesting maneuvers and bottlenecks. The vastness of parameters made selecting a small diverse set of instances extremely challenging. We discuss tools to deal with instance selection in Section 10.4 and the generator is described in Subsection 13.2.3. While the instance selection took less than a day each for the first two years, it took around two weeks for the third competition. The time investment for the currently-running fourth competition has even been higher, because many random instances proved to be surprisingly easy to solve, so the generation of interesting instances required more care.

10.3.4. Support

During the competition, we need to provide support to the participants. A special email address has been set up for this purpose. We received around 10 to 30 inquiries and requests per competition. During the first two competitions, this involved many smaller issues regarding problems with the webpage, which we optimized afterward. Many mails also requested specific details either regarding rules or the webpage. The most complicated requests required support analyzing why the submission has been rejected. This was also the reason why we made the verifier public after the first competition. Often the reason for rejection of a submission was either due to geometric edge cases or misunderstood rules. Sometimes security mechanisms, e.g., against malicious archives, rejected files and needed an adjustment or manual override. Especially due to the last type of case, support required around 5 days per competition, but this decreased with more experience.

10.3.5. Maintenance

Server maintenance is probably the most time-consuming and stressful element of hosting CG:SHOP. There are many components that have to work correctly with each other and need careful configuration. After having had some very bad experiences with inconsistent updates and race-conditions after reboots, we migrated the system to Docker containers.

Many of these problems most likely arose from problematic or outdated configurations. Unfortunately, setting up all parameters correctly can require considerable knowledge or research. The unified, preconfigured, and isolated environment by Docker allows us to set up such a system with considerably less knowledge, and it eases looking up issues. Unexpected problems can still arise, for example we had to spend quite a bit of time fixing a problem with Docker and the used NFS. Setting up servers, and keeping them up to date can be a serious and easily-overlooked amount of work, estimated at around three weeks per year.

10.3.6. Analysis

If we want to provide more detailed information than just who achieved the highest score, we need to do some data analysis. This requires insight into the instances and problem, especially the ability to partition them into different classes to find strengths and weaknesses of the submitted solutions. Such analysis should not only be performed at the end, but also during the competition to detect problems early. An example of this is in the CG:SHOP 2020, where we noticed that the initial instances were too simple and the leading team's submissions nearly identical. While the first competition was easy to analyze, the last two took multiple days for the intermediate and final analyses.

10.4. Instance Selection

One problem we are faced with for every competition is the selection of instances. On one hand, we need a reasonably small set, because many participants are students who only have a single workstation. Solving thousands of instances is beyond their resources. On the other hand, we need a diverse set because some instances are prone to being easy to solve, but we do not know which ones in advance. For NP-hard problems, there often is some instance size limit for which we can still solve the instances to optimality and shortly afterward, instances become exponentially harder. We can try to find the order of magnitude by quickly implementing a MIP, CP, or SAT-formulation, but there is always the chance that we overlooked something. For some NP-hard problems, one can solve even very large instances by local heuristics because random instances simply do not contain any of the hard elements which are sometimes very structured, e.g., by collinearity. An example is the MINIMUM-WEIGHT TRIANGULATION which has been proved to be NP-hard by Mulzer and Rote [259], but instances with 30 000 000 points have been solved to provable optimality within 4 min by Haas [260]. The instances become more difficult if they contain point sets that form a regular n -gon with a single point in the center, but such cases are rarely created by random instance generators if they are not specially designed for this purpose.

In the following, we describe a technique that tries to distill the ideas used in CG:SHOP. These steps are only a rough guideline and still require trial-and-error and manual fine-tuning. Especially selecting the features for comparing instances usually cannot be done without the corresponding knowledge and educated guesses. On the other hand, if we are able to directly engineer some very strong features, we may not

even need the following steps anymore but we can select the instances directly.

1. We try to estimate the correct order of magnitude for instance sizes and other necessary properties. We can usually quickly devise an IP, CP, or SAT-formulation that can be used to find the lower bound, i.e., which instances are too easy. The upper bound is much harder to estimate. Here we can aim for a logarithmic distribution that contains only a few very large instances, as we do not want to discourage participants with too many too hard instances. At the same time, we want some very hard instances as tie-breaker in case we overestimated the difficulty of the problem.
2. We create instances using various random generators or, if available, existing (real-world) instances. We use a diverse set of parameters for the generators, and do not worry about too many similar instances yet. The individual parameters may be interesting features for the final selection and should be saved with the instance.
3. We devise a set of general instance features. For a graph, the number of vertices and edges is such a feature.
4. If we have some solvers already available, we add their solutions, bounds, and runtime as features. If the solver only yields solutions for simple instances, but none for larger instances, this may also be worth a feature. Such information can help to identify hard or interesting instances. If all solvers take a long time, this is probably a hard instance. If the standard deviation of the objective values is high, this is probably an interesting instance that requires some global optimization.
5. We create composite features that combine multiple features into one, or normalize them. For example, the deviation of the objective values mentioned in the previous step generally increase with larger objective values. We can normalize them via dividing by the best or mean objective value. If we combine multiple features into one, as we did with the standard deviation of the objective values, we may directly delete the original features if they only add little additional information. We try to combine and reduce as many features as possible because this eases the next steps.
6. We need to take care of incomplete features and missing values, before we can compare the instances. We may need to split the instance set if the features of larger instance sets are not comparable. This might be the case if they have been generated by different generators and the parameters prove to be important features. In some cases, we may be able to extract this important feature from the instance itself. If not, we can simply decide on a specific number of instances from each generator and perform the selection individually. We can also try to fill the missing values by interpolation, which can be done simply by the mean value or by machine learning based on the other features as a more complex route.
7. We look into the distributions and correlations of the features. During this, we may notice some irregularities that allow us to remove or combine some further features. We try to remove all features that do not look promising such that we are only left with around 10 features. Pair plots are also a great option to do so, see Figure 10.7. If there are too many features, we may select a subset of these and perform multiple comparisons.

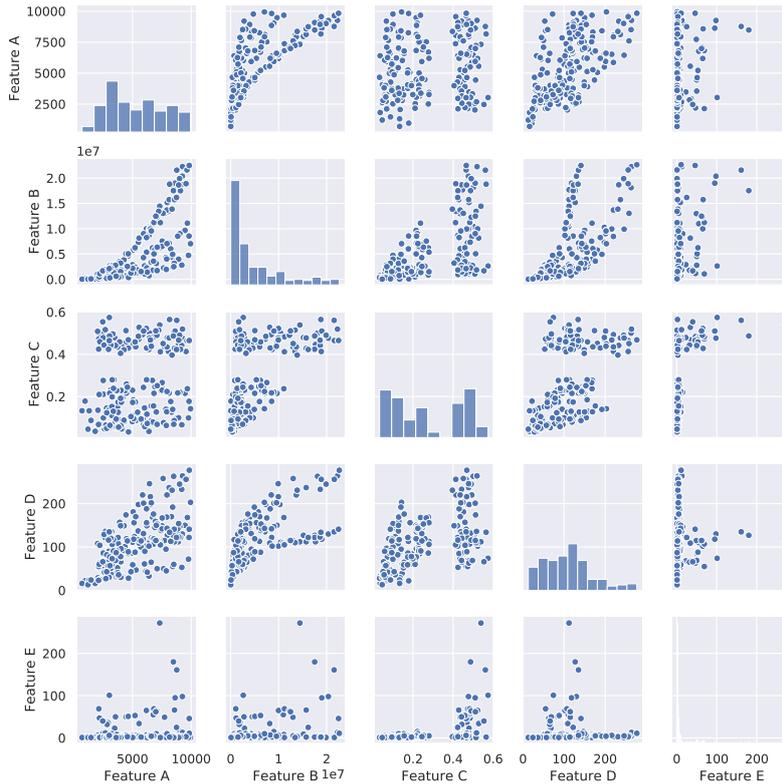


Figure 10.7.: Features of instances (based on real data) shown with correlations in a pair plot. We can directly see some non-linear correlations and distributions.

8. We scale the features such that we can compute a useful distance on the values. Using a standard scaler that moves the mean to zero and scales the standard deviation to one is a great option. This provides outliers with a larger distance to the other instances; outliers can be interesting instances. Otherwise, we can use a simple min-max-scaling. Min-max-scaling scales the values into the range of 0 to 1, which can lose its expression in the presence of outliers. The best option depends on the concrete feature.
9. We remove correlations by using a principal component analysis (PCA) and by performing a dimensionality reduction. The number of remaining dimensions should be as small as possible, as long as the induced error does not become too high. This again needs some fine-tuning. We can check the error either by common metrics or simply by retransforming the features and comparing the pair plots, see Figure 10.8. Having only few, e.g., three or four, dimensions with low correlation by which to compute the distance makes the selection much more robust. Due to the infamous *curse of dimensionality*, the feature space gets much sparser with every additional dimension. In a sparse feature space, the distance between any two instances is large and it is harder to perform a proper selection of distant instances.
10. Perform a clustering with *kMeans* of the instances in the new space, see Figure 10.9. The number of clusters should equal the desired number of instances. From each cluster, select a random instance. The clustering ensures we select instances from different areas of the feature space. We can also select the center of the cluster instead of a random instance, but the center will be more average than a random instance.

Figure 10.8.: Using standard scaling and a principal component analysis, we can reduce the data to three dimensions by removing linear correlations while only introducing a small amount of error. The right number of dimensions can differ with different instances, but the smallest possible should be chosen in order to have a better distance function. Note that features with strong but unimportant and uncorrelated values can harm this step. The new coordinates are in orange, the original in blue.

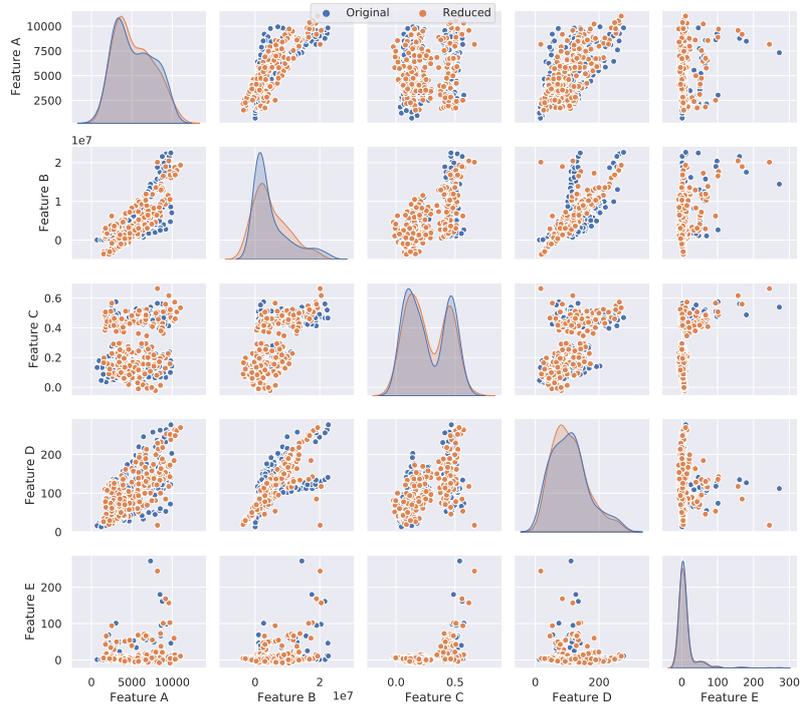
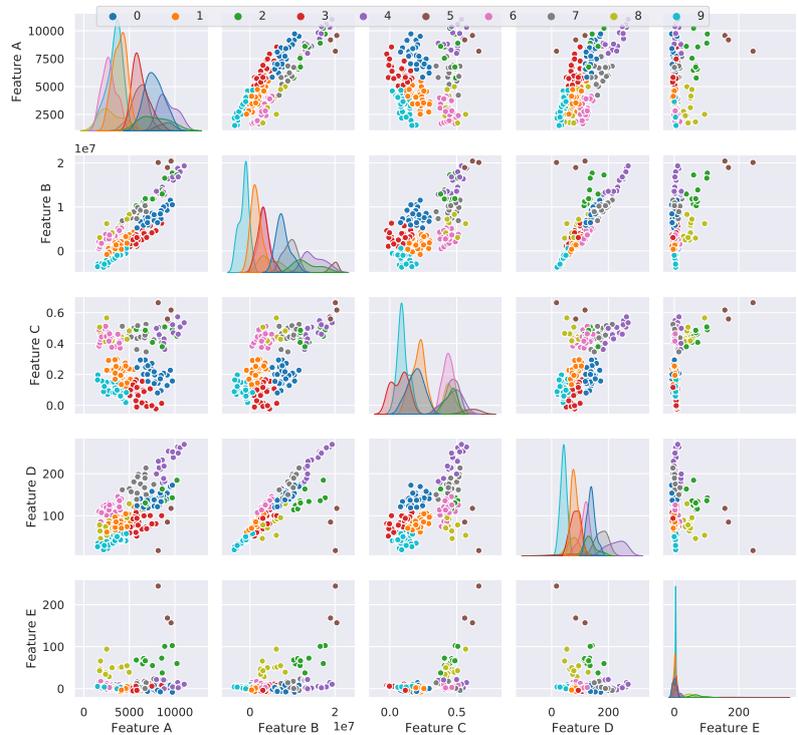


Figure 10.9.: The clustering is based on the Euclidean distance in the reduced space. Each of the 10 clusters is shown in its own color. This is also great for exploration, as we can see that the brown instances are extreme in multiple features. Selecting one instance per cluster yields a diverse set.



11. We check the distribution of the instances and fine-tune the process, if needed. We may also split this process for different classes of instances, e.g., for small, medium, and large instances. If the classes become too small, this process will of course not be much better than random; but in this case, we can also select the instances by hand.

Great tools in Python (2021) for performing these steps are:

- ▶ *pandas* for data management and manipulation.
- ▶ *scikit-learn* for scaling and dimensionality reduction.
- ▶ *seaborn* for nice and simple plots based on *matplotlib*.
- ▶ *jupyter* for interactive development, ideal for parameter-tuning. Also great for documentation and sharing.

Future Work 10.1.

Can we develop a toolkit that combines all of the important techniques and provides a streamlined interface? Additionally, some interactive visualization and manual selection tools may be very helpful.

The described approach drew inspiration from the works of Smith-Miles et al. [261–267], who are working on the instance feature spaces, e.g., for performance prediction of algorithms and the ALGORITHM SELECTION PROBLEM [268]. One idea is to use evolutionary algorithms to evolve interesting instances. This approach is also used to generate a diverse instance set for GRAPH COLORING [267], but the focus is on discovering novel instances rather than selecting a diverse set of limited size. A library for instance space analysis is provided by MATILDA [269]. The primary differences between these techniques and our approach are:

- ▶ We are interested in actual instances and not the feature space.
- ▶ We do not have the actual algorithms. In fact, the algorithms will evolve to fit the instances and not the other way around.

However, intervening the instance feature space and the instance generation is an approach that also has potential for benchmark creation. This technique was not necessary for use, as we did not (yet) have issues with a too sparse instance feature space.

10.5. Lessons Learned

Before discussing the individual competitions in detail, let us summarize some lessons learned:

- ▶ Setting up a minimal server for processing uploads is reasonably easy and quick, thanks to generic frameworks. Additional (optional) elements that need a complicated integration or need to be implemented from scratch, take up most of the development time.
- ▶ The complexity of instance selection and scoring can vary strongly for different problems.

- ▶ Most of the submissions happen shortly before the deadline. While the previous years always had some teams that started out early and strong, the top three teams still usually changed position during the last couple of day(s).
- ▶ Some teams struggle with the upload because their files are too large or their connections too slow. An option is to allow the submission of instances by email with a hash of the compressed archive. The actual archive can be sent later via a link to a cloud provider. This provides a robust alternative to the regular submission, while still allowing a fair evaluation. It is important to note that the date of an email can easily be manipulated; instead the time of the reception should be used.
- ▶ It is important to be predictable and transparent. Changing the rules during a game does not feel fair to the participants, even if the change benefits them. Depending on the magnitude of the change, participants might be discouraged to the extent that they put forth less effort in the competition, out of fear that their effort might be in vain following a possible change of rules. Even providing an auxiliary tool after the start of the competition can create discontent among those who have already spent energy developing such a tool themselves.

This chapter gives an overview of the theoretical and practical aspects of finding a simple polygon of minimum (MIN-AREA) or maximum (MAX-AREA) possible area for a given set of n points in the plane. Both problems are known to be NP-hard and were the subject of the 2019 Computational Geometry Challenge, which presented the quest of finding good solutions to more than 200 instances, ranging from $n = 10$ all the way to $n = 1\,000\,000$.

11.1 Introduction	243
11.1.1 Challenge Problem . . .	243
11.1.2 Related Work	244
11.1.3 Outcomes	245
11.2 Pick’s Theorem and Integrality	246
11.3 Approximation	247
11.4 Contest and Outcomes .	248
11.4.1 Instances	248
11.4.2 Evaluation	249
11.4.3 Results	249
11.5 Conclusions	254

11.1. Introduction

The “CG:SHOP Challenge” (Computational Geometry: Solving Hard Optimization Problems) originated as a workshop at the 2019 Computational Geometry Week (CG Week) in Portland, Oregon in June, 2019. The goal was to conduct a computational challenge competition that focused attention on a specific hard geometric optimization problem, encouraging researchers to devise and implement solution methods that could be compared scientifically based on how well they performed on a database of instances. While much of computational geometry research has targeted theoretical research, often seeking provable approximation algorithms for NP-hard optimization problems, the goal of the CG Challenge was to set the metric of success based on computational results on a specific set of benchmark geometric instances. The 2019 CG Challenge focused on the problem of computing simple polygons whose vertices were a given set of points in the plane.

11.1.1. Challenge Problem

The TRAVELING SALESMAN PROBLEM (TSP) is one of the classic optimization problems: For a given set of locations and pairwise distances, find a shortest roundtrip that visits each position exactly once and returns to the start. In a geometric setting, in which locations correspond to a given set P of n points in the plane, and distances between points are induced by the Euclidean metric, it is a straightforward consequence of the triangle inequality that an optimal tour corresponds to a simple polygon \mathcal{P} with vertex set P , such that \mathcal{P} has minimum total perimeter length.

This geometric motivation makes it natural to consider simple polygons with a given set of vertices that minimize another basic geometric measure: the enclosed area. This was considered in the past in the context of surface reconstruction, e.g., by O’Rourke [270]; as sketched in Section 11.2, there is also a close connection to point separation, which has gained importance in Artificial Intelligence. The same context has also raised interest in the corresponding maximization problem: For a given set P of n points in the plane, find a simple polygon \mathcal{P} with vertex set P of maximum possible

* A preliminary version [11] of this chapter was published on arXiv. Many thank to Erik Demaine, Sándor Fekete, Phillip Keldenich, and Joseph Mitchell for all their contributions.

area. In the following, we refer to these two problems as MIN-AREA and MAX-AREA, respectively.

Problem 11.1.1 (MIN-AREA and MAX-AREA)

Given: A set P of n points in the plane.

Goal: A simple polygon with vertex set P of minimum (MIN-AREA) or maximum (MAX-AREA) possible enclosed area.

There are a number of features that make these problems suitable for optimization challenges, based on notable similarities to and differences from the TSP. As shown by Fekete [271–273], both MIN-AREA and MAX-AREA are NP-hard; however, while membership in NP for the Euclidean TSP is a long-standing, famous open problem (e.g., Problem #33 in The Open Problems Project [28]), it is straightforward for area optimization, so issues of numerical stability and checking validity of solutions do not come into play. On the other hand, edges in a polygon with small area need not be short, as shown in Figure 11.1. This makes it challenging to restrict potential neighbors of a point in a good polygonization, increasing the difficulty of employing local search methods for efficient algorithms. This explains an apparent practical distinction to the TSP: while provably optimal solutions for TSP benchmark instances of considerable size have been known for a while, exact methods for area optimization appear more elusive.

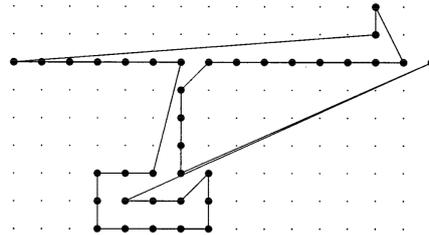


Figure 11.1: A point set P and its only minimum-area polygon \mathcal{P} , containing several long edges. (Example from [271, 273].)

11.1.2. Related Work

History and Background

The origins of the problem of finding area-optimal polygons can be traced back at least to the early days of Computational Geometry (O’Rourke [270] in 1980). Resolving the complexity was posed by Suri in 1989 at CCCG, and restated in a different context by Mitchell [274] and Mitchell and Suri [275].

Studying the set of possible polygonizations is of great interest in various applications [276–280]. Auer and Held [281] gave methods for generating random polygonizations. O’Rourke and Suri [282] raised the question of estimating the number of polygonizations as a function of the number n of points. At this point, this is known to lie between 4.642^n [283] and 56^n [284].

Complexity

Fekete [271, 273] gave a proof of NP-completeness for both problems, based on a reduction from HAMILTONICITY OF PLANAR CUBIC DIRECTED GRAPHS, and generalized this proof to higher dimensions: For any fixed $1 \leq k \leq d$ and $2 \leq d$, it is NP-hard to find the polyhedron with k -dimensional faces of minimum total volume for given vertices in d -dimensional space. He also showed that no polynomial-time approximation scheme (PTAS) exists for MIN-AREA and presented a $\frac{1}{2}$ -approximation algorithm for MAX-AREA [271, 272]. Moreover, he proved that the NP-hardness of the minimization problem also applies for higher dimensions. More specifically he showed that for given $1 \leq k \leq d$ and $2 \leq d$ it is NP-hard to find the minimal volume polyhedron with k -dimensional faces for given vertices.

Heuristics

Recent work was mainly focused on finding new heuristics for both MIN-AREA and MAX-AREA. Taranilla et al. [285] proposed three different heuristics. Peethambaran et al. [286, 287] proposed randomized and greedy algorithms for MIN-AREA and d -dimensional variants of both problems.

Other Challenges

The Open Problems Project (TOPP), maintained by Demaine, Mitchell and O'Rourke [28], is a library of long-standing unsolved problems. On the more practical side, there have been different efforts, based on benchmark libraries, such as the TSPLIB [198]. Since 1990, the DIMACS implementation challenges have addressed questions of determining realistic algorithm performance where worst-case analysis is overly pessimistic and probabilistic models are too unrealistic. Since 1994, the Graph Drawing (GD) community has held annual contests in conjunction with its annual symposium to monitor and challenge the current state of the graph-drawing technology and to stimulate new research directions for graph layout algorithm. More recently, a variety of implementation challenges have gained traction in the world of programming and optimization, but not yet in the field of Computational Geometry.

11.1.3. Outcomes

The contest generated a large number of contributions, both for MIN-AREA and MAX-AREA. In the aftermath, the top teams were invited to describe their methods in detailed papers, which form the substance of this special issue.

- ▶ Julien Lepagnot, Laurent Moalic, Dominique Schmitt: Optimal area polygonization by triangulation and ray-tracing [288]
- ▶ Loïc Crombez, Guilherme D. da Fonseca, Yan Gerard: Greedy and Local Search Solutions to the Minimum and Maximum Area [289]
- ▶ Nir Goren, Efi Fogel, Dan Halperin: Area-optimal polygonization using simulated annealing [290]

- ▶ Günther Eder, Martin Held, Steinpor Jasonarson, Philipp Mayer, Peter Palfrader: 2-Opt moves and flips for area-optimal polygonizations [291]
- ▶ Natanael Ramos, Rai Caetan de Jesus, Pedro de Rezende, Cid de Souza, Fabio Luiz Usberti: Heuristics for area optimal polygonizations [292]

In addition, there is one paper focusing on exact methods for computing provably optimal solutions.

- ▶ Sándor P. Fekete, Andreas Haas, Phillip Keldenich, Michael Perk, Arne Schmidt: Computing area-optimal simple polygonization [293]

In the rest of this survey paper, we provide a discussion of specific aspects of mathematical connections between area optimization and grid points (Section 11.2), approximation algorithms (Section 11.3), and an overview of contest results (Section 11.4).

11.2. Pick’s Theorem and Integrality

For a simple polygon \mathcal{P} with n vertices, computing the Euclidean length of its perimeter involves evaluating a sum of square roots, for which membership in NP is a long-standing open problem [28]. This differs from computing the area of \mathcal{P} , which can be evaluated quite efficiently, e. g., see O’Rourke [294]. An elegant combinatorial answer is given by Pick’s theorem. (This also implies benign objective values, in particular for vertices whose coordinates are all even numbers, as chosen in the contest.)

Theorem 11.2.1 (Pick [295]) *Let \mathcal{P} be a simple polygon with integer vertices; let $i(\mathcal{P})$ be the number of grid points contained in the interior of \mathcal{P} , and let $b(\mathcal{P})$ be the number of grid points on the boundary of \mathcal{P} . Then*

$$\text{Area}(\mathcal{P}) = \frac{1}{2}b(\mathcal{P}) + i(\mathcal{P}) - 1.$$

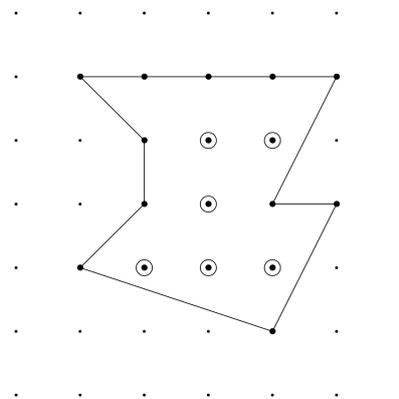


Figure 11.2.: Pick’s theorem: A simple grid polygon \mathcal{P} with $b(\mathcal{P})$ grid points on its boundary and $i(\mathcal{P})$ grid points in its interior has area $\frac{1}{2}b(\mathcal{P}) + i(\mathcal{P}) - 1$. (Here shown for $b(\mathcal{P}) = 11$ and $i(\mathcal{P}) = 6$.)

There are several elegant ways to prove Pick’s theorem, three of which can be found in [296–298]. For a discussion of alternative approaches see the article by Niven and Zuckermann [299]. There are numerous

generalizations to other than the orthogonal grid, e.g., by Ren and Reay [300]; see Reeve [301] for a generalization to higher dimensions.

Pick's theorem also provides a bridge to issues of point separation: Maximizing the enclosed area amounts to finding a simple polygon that captures as many additional grid points as possible, while minimizing the area corresponds to excluding as many as possible. As the n given points must lie on the boundary of \mathcal{P} , and only points within the convex hull of P come into play, we get the following lower and upper bounds for the area.

Theorem 11.2.2 *Let P be a set of n points in the plane that all have integer coordinates. Let $h_i(P)$ denote the number of points of the integer grid that are not contained in P and strictly inside the convex hull, and let $h_b(P)$ be the number of grid points not in P that are on the boundary of the convex hull.*

Then for any simple polygon \mathcal{P} on the vertex set P , we have

$$\frac{n}{2} - 1 \leq \text{Area}(\mathcal{P}) \leq \frac{n}{2} + \frac{h_b(P)}{2} + h_i(P) - 1.$$

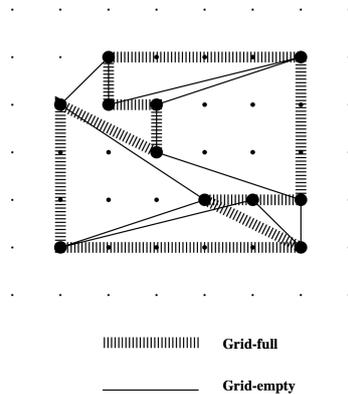


Figure 11.3: Lower and upper bounds on polygon area, implied by Pick's theorem: For a set P of grid points (bold), shown a *grid-empty* simple grid polygon that contains only the given points and a *grid-full* polygon that contains all grid points of the convex hull to the maximum possible extent.

See Figure 11.3 for an illustration. Deciding whether either of these bounds can be met is already NP-complete [273]. At the same time, they motivate using the area of the convex hull as a reference, which was used in the contest.

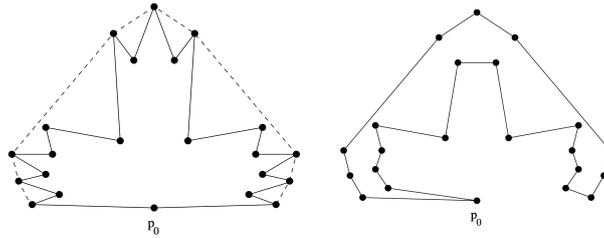
11.3. Approximation

Using the area of the convex hull as an upper bound can also be used for approximating MAX-AREA.

Theorem 11.3.1 ([271]) *Let P be a set of n points in the plane. We can determine a simple polygon \mathcal{P} on P that has area larger than $\frac{1}{2}\text{Area}(P)$, where $\text{Area}(P)$ denotes the area of $\text{conv}(P)$. This can be done in time $O(n \log n)$.*

Proof. Let p_0 be a point on the convex hull of P . In time $O(n \log n)$, sort the points p_i of P by the slope of the lines $l(p_0, p_i)$, such that the neighbors of p_0 on the convex hull are the first and the last point, respectively. If

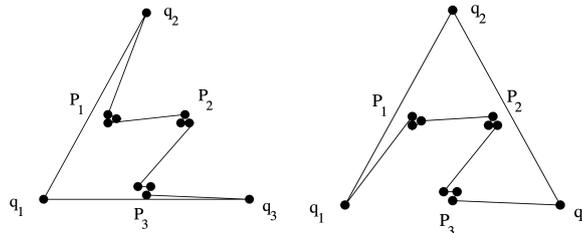
Figure 11.4: A $\frac{1}{2}$ -approximation for MAX-AREA: A star-shaped polygon \mathcal{P}_1 around a hull point p_0 (left) and a second simple polygon \mathcal{P}_2 covering the rest of the convex hull.



there is a set of points for which the slope is the same, break the tie by ordering them in increasing distance from p_0 , except when those points have the smallest of all slopes, in which case we take them in order of decreasing distance from p_0 . Connecting the points p_i in this order yields a simple polygon \mathcal{P}_1 on P .

If $Area(\mathcal{P}_1) > \frac{1}{2}Area(P)$, we are done. Suppose this is not the case. Then the set $\mathcal{Q} := conv(P) \setminus \mathcal{P}_1$ has area at least $\frac{1}{2}Area(P)$. Now it is not hard to see that there is a simple polygon \mathcal{P}_2 that contains \mathcal{Q} , implying the claim. \square

Figure 11.5: The approximation factor of $\frac{1}{2}$ for the algorithm is tight.



As shown in Figure 11.5, $\frac{1}{2}$ is a tight bound for this approach, even if all possible choices for p_0 are tested. Moreover, it was shown in [271] that it is NP-complete to decide whether there is a simple polygon that contains strictly more than $\frac{2}{3}$ of the area of the convex hull;

At this time, no constant-factor approximation algorithm for MIN-AREA is known, hinting at a higher level of difficulty of the minimization problem. It may well be the case that no such approximation can be computed in polynomial time.

11.4. Contest and Outcomes

The 2019 Challenge was well received, with 28 teams from all over the world and a range of different scientific areas competing; participation was open to anyone. The contest itself was run through a dedicated server at TU Braunschweig, hosted at <https://cgshop.ibr.cs.tu-bs.de/competition/cg-shop-2019/>. It opened on February 28, 2019, and closed on May 31, 2019.

11.4.1. Instances

The contest started with a total of 247 benchmark instances, as follows. Each of these instances consisted of n points in the plane with integer coordinates. For $n \in 10, 15, 20, 25, 30, 35, 40, 45, 50, 60, 70, 80, 90, 100, 200$,

300, 400, 500, 600, 700, 800, 900, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10 000, 20 000, 30 000, 40 000, 50 000, 60 000, 70 000, 80 000, 90 000, 100 000}, there were six instances each. In addition, there is one instance of size $n = 1\,000\,000$.

The instances were of three different types; see Fekete et al. [21] for a more detailed description of how to generate benchmark instances through illumination maps.

- ▶ **uniform:** uniformly distributed at random from a square.
- ▶ **edge:** randomly generated according to the distribution of the rate of change (the “edges”) from various images.
- ▶ **illumination:** randomly generated according to the distribution of brightness of an image (such as an illumination map).

11.4.2. Evaluation

The comparison between different teams was based on an overall *score*. For each instance, this score is the ratio given by the achieved area divided by the area of the convex hull; thus, the score is a number between 0 and 1. The total score achieved by each team was the sum of all 247 individual instance scores. Feasibility of submitted solutions was checked at the time of upload; for instances without a feasible solution, a default score of 1 (for minimization) or 0 (for maximization) was used. For multiple submissions by the same team, only the best feasible solution submitted was used to compute the score. In case of ties, the tiebreaker was set to be the date/time a specific score was obtained. This turned out not to be necessary.

11.4.3. Results

In the end, the top 10 in the leaderboard looked as shown in Table 11.1; note that according to the scoring function, a lower score is better for MIN-AREA, while a higher score is better for MAX-AREA. The progress over time of each team’s score can be seen in Figure 11.6a (for MIN-AREA) and Figure 11.6b (for MAX-AREA).

Table 11.1: The top of the final leaderboard. Shown are the scores in both categories, along with the achieved average percentages of the convex hull of points. Teams CGA and UNICAMP were overall tied, according to different positions for MIN-AREA and MAX-AREA, as were mperk and AQ_PG. The teams mperk (Michael Perk, TU Braunschweig) and zhengdw (David Zheng, University of British Columbia) were both individual, first-year Masters students.

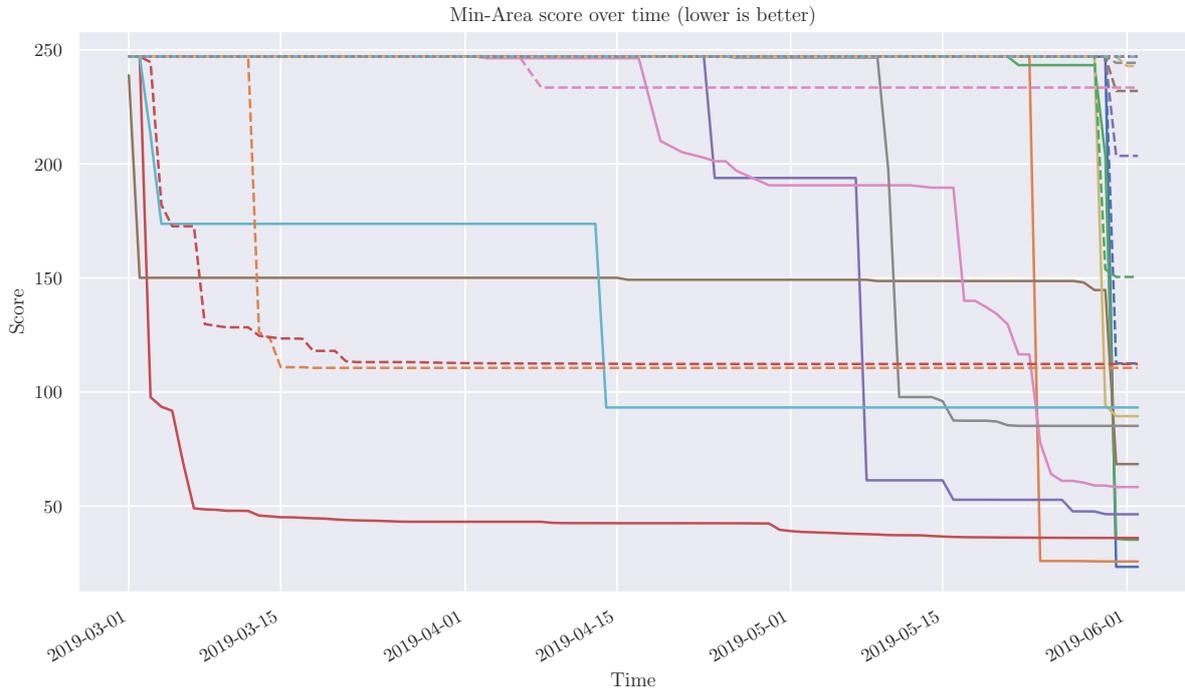
Rk.	Team	Score (Min)	Score (Max)	# best (unique)		# best (unique)	
				Min	sols.	Max	sols.
1	OMEGA/Mulhouse (FR)	23.393 (9.47%)	227.247 (92.00%)	223	(181)	184	(138)
2	lcrombez/Clermont (FR)	25.751 (10.43%)	226.691 (91.78%)	66	(23)	109	(4)
3	cgl@tau/Tel Aviv (IS)	35.289 (14.29%)	206.612 (83.65%)	13	(0)	12	(0)
4	CGA/Salzburg (AU)	36.069 (14.60%)	197.568 (79.99%)	26	(0)	23	(0)
4	UNICAMP/Campinas (BR)	46.432 (18.80%)	201.839 (81.72%)	3	(0)	3	(0)
6	mperk/Braunschweig (GE)	68.431 (27.70%)	191.483 (77.52%)	24	(0)	23	(0)
6	L’Aquila-Perugia (IT)	57.373 (23.23%)	179.752 (72.77%)	19	(0)	18	(0)
8	Stony Brook (US)	85.179 (34.49%)	162.031 (65.60%)	1	(0)	1	(0)
9	zhengdw (CA)	89.437 (36.21%)	154.723 (62.64%)	0	(0)	1	(0)
10	TGP/Eindhoven (NL)	112.561 (45.57%)	154.548 (62.57%)	18	(0)	26	(0)

The top 5 finishers were invited for contributions to this special issue, as follows.

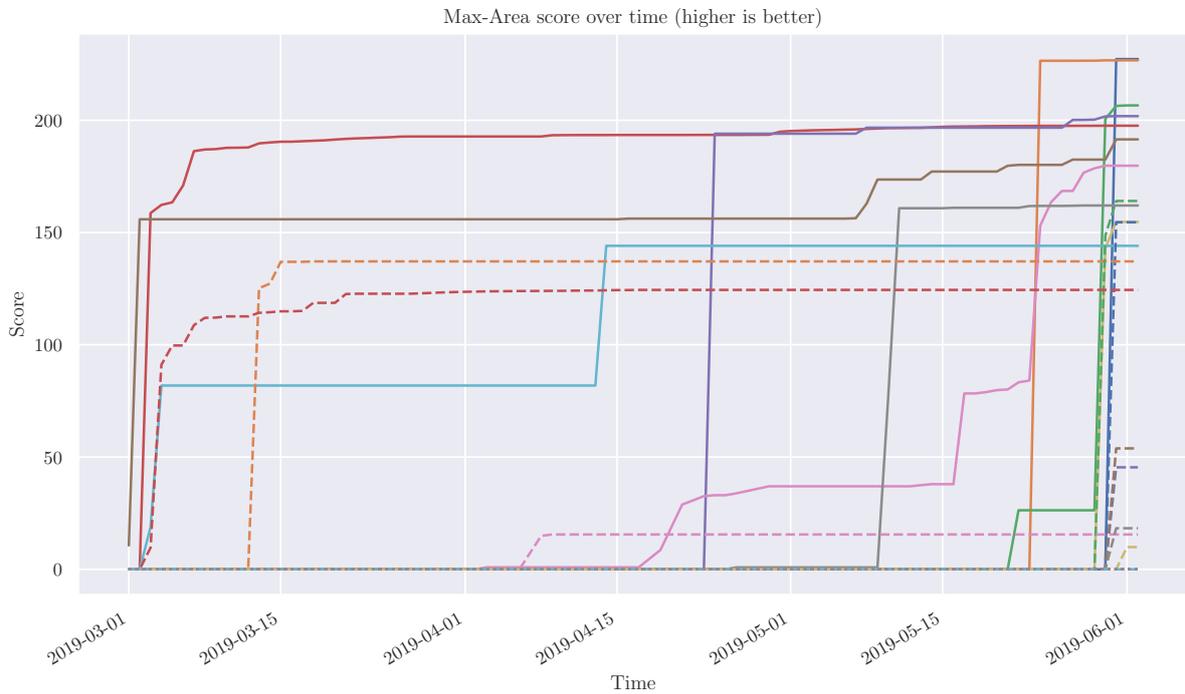
1. Team OMEGA/Mulhouse (France): Julien Lepagnot, Laurent Moalic, and Dominique Schmitt [288].
2. Team IcmombeZ/Clermont Auvergne(France): Loïc Crombez, Guilherme D. da Fonseca, and Yan Gerard [289].
3. Team cgl@tau/Tel Aviv (Israel): Nir Goren, Efi Fogel, and Dan Halperin [290].
4. Team CGA/Salzburg (Austria): Günther Eder, Martin Held, Steinthor Jasonarson, Philipp Mayer, and Peter Palfrader [291].
5. Team UNICAMP/Campinas (Brazil): Natanael Ramos, Rai Caetan de Jesus, Pedro de Rezende, Cid de Souza, and Fabio Luiz Usberti [292].

Figure 11.6a shows the development of total scores over time for MIN-AREA and all teams; it can be seen that OMEGA passed IcmombeZ shortly before the deadline, with cgl@tau just squeezing by CGA. A similar and even tighter outcome between OMEGA and IcmombeZ for MAX-AREA can be seen in Figure 11.6b; for that problem, cgl@tau also placed third, but UNICAMP managed to beat out CGA for fourth place. Thus, the ranking was consistent for both MIN-AREA and MAX-AREA, except for Campinas doing better than Salzburg in MAX-AREA, so they both shared fourth place. All five teams engineered their solutions with the use of a variety of specific tools. Details of their methods and the engineering decisions they made are given in their respective papers.

Figure 11.7a shows the spread of results for all MIN-AREA instances; it can be seen that there is a strong deviation over all instance sizes, even for the small ones. This is surprising as one would expect simple heuristics to perform reasonably well for small instances; however, it appears that even for very small instances with 10 points of MIN-AREA, more advanced ideas are necessary to achieve good results. (See the contribution by Fekete et al. [293] for a detailed study of exact methods.) At around 5000 points, the mean score drops visibly; the best teams are able to obtain nearly the same score for all instance sizes above 50 points. Surprisingly, the effect is stronger for uniform point sets. However, when normalizing the scores in Figure 11.8a we see that this actually seems to be an effect of the bound quality. When using the best available solution as reference, the image-based instances show more of a struggle for the teams to keep up. A similar overview for MAX-AREA is given in Figure 11.7b. Here the deviation is very small for the small instances, showing that all teams were able to obtain reasonably good solutions for small instances. The deviation increases with the problem size until around 300 points, after which it remains homogenous but smaller than for MIN-AREA. Like for MIN-AREA, the best teams obtained nearly equal scores for all instance sizes, while the mean score drops visibly after 900 points. Again, the effect is stronger for uniform point sets, but also remains so after using the best solution as reference in Figure 11.8b.

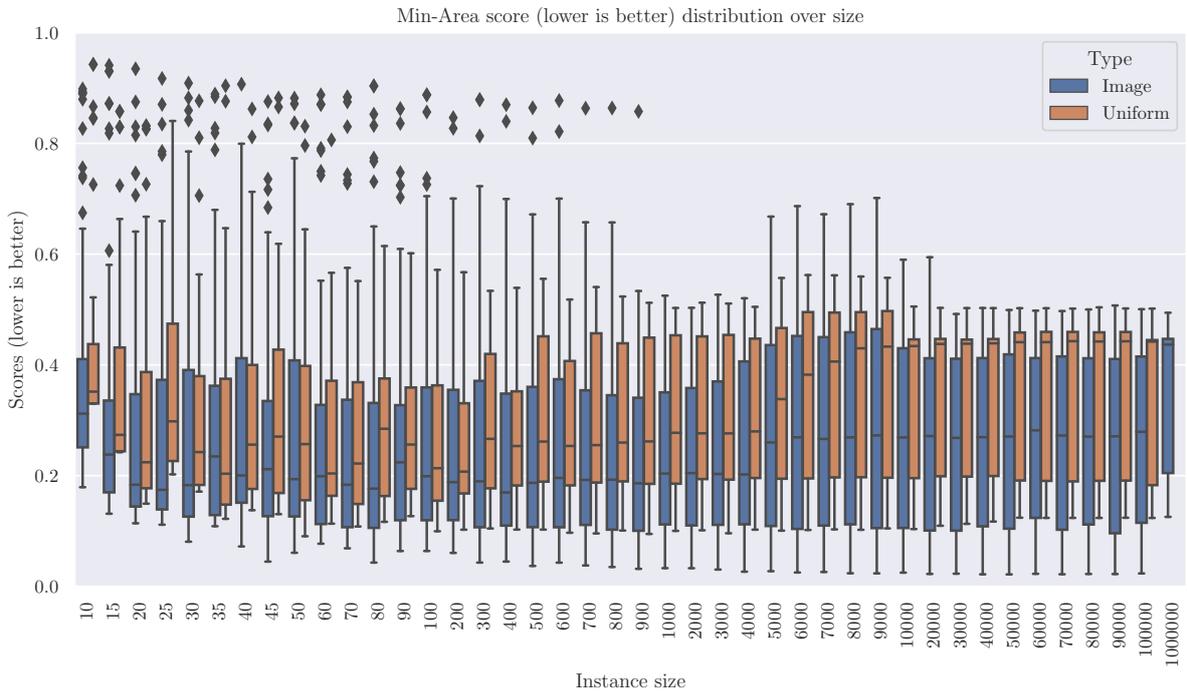


(a) MIN-AREA.

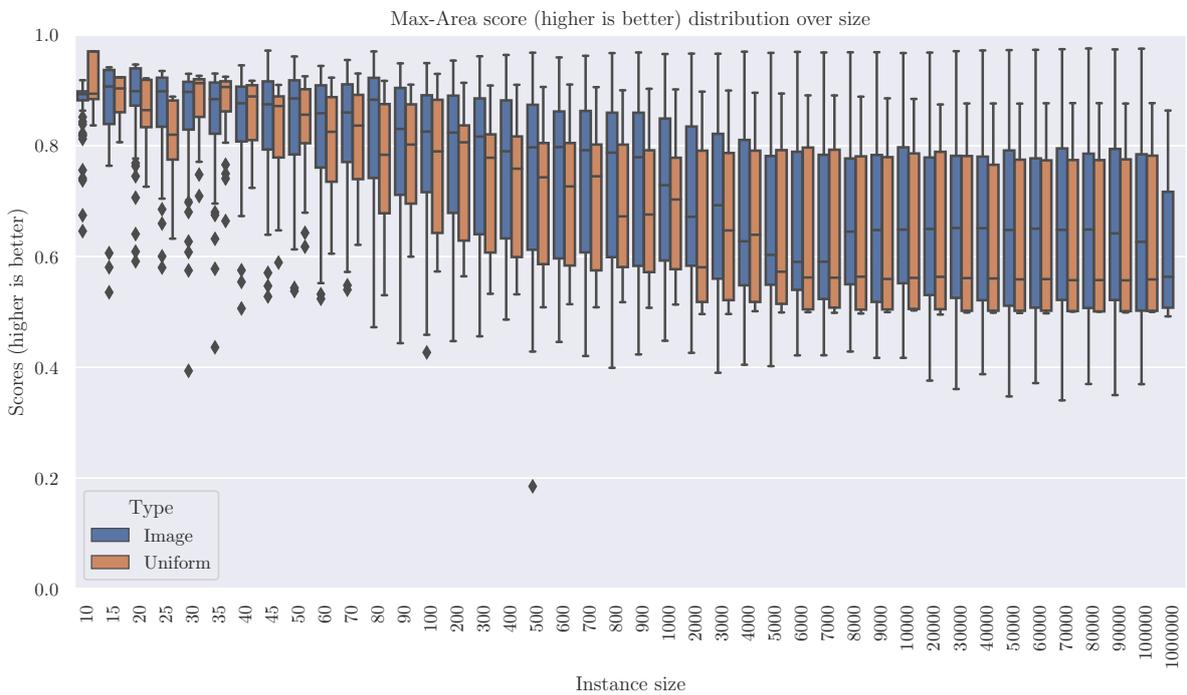


(b) MAX-AREA.

Figure 11.6.: Total score over time for MIN-AREA and MAX-AREA of the participating teams. The team names of the ten leading teams can be deduced from Table 11.1. We see teams that started early and strong and teams that just started to submit solutions shortly before the deadline. The last days had especially many strong jumps.

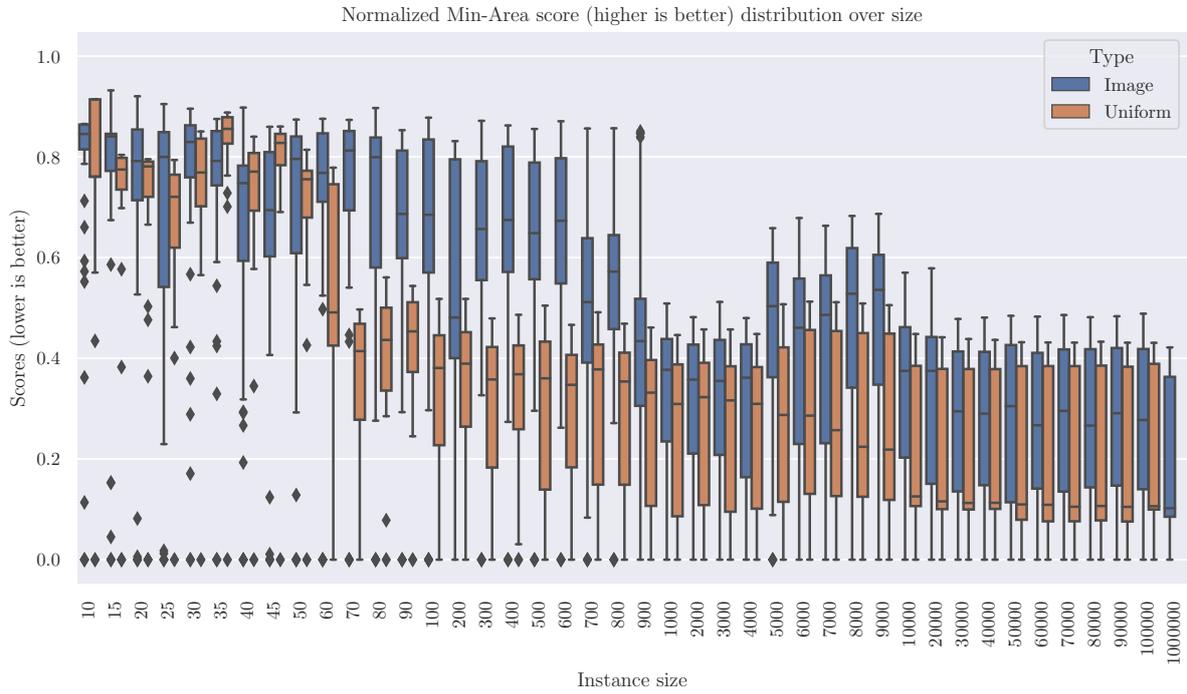


(a) MIN-AREA.

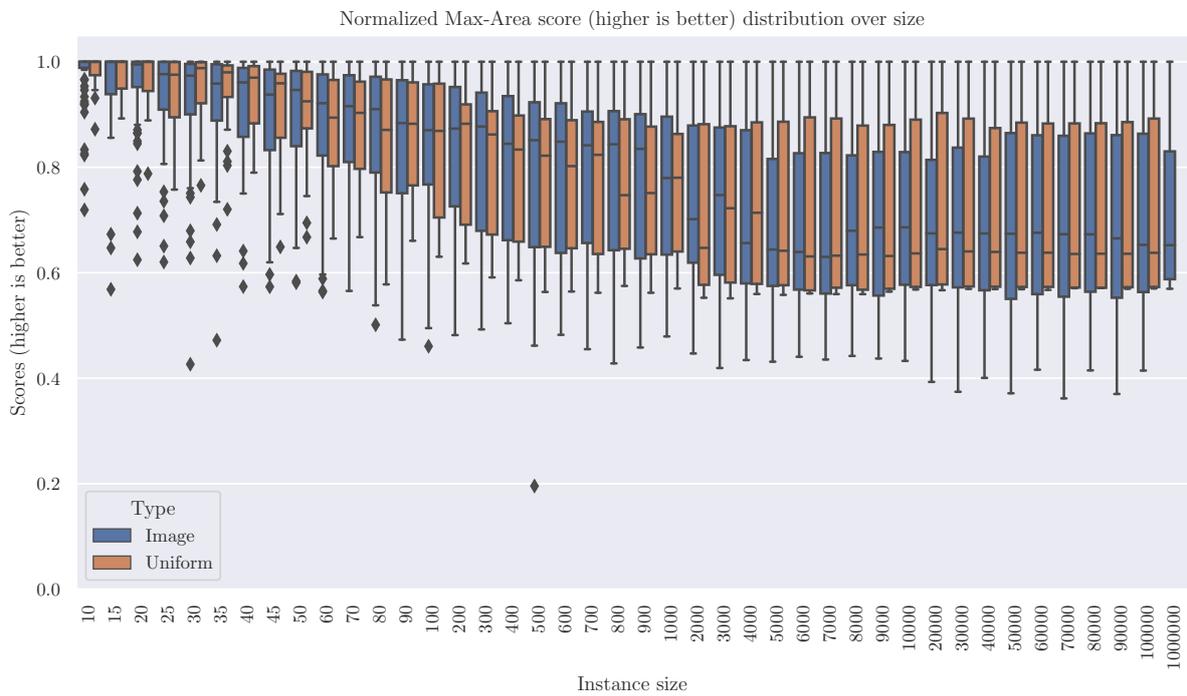


(b) MAX-AREA.

Figure 11.7.: Distribution of best scores over all instances. The middle line shows the median. The boxes show the quartiles and the whiskers the min and max (except possibly outliers as additional dots). If there were multiple instances per size, the mean score for those is used. Only submitted instances are considered such that larger instances are potentially based on less data points. For MAX-AREA The image based instances have a higher variance but also a higher median score.



(a) MIN-AREA.



(b) MAX-AREA.

Figure 11.8.: Normalized score distribution by putting it in relation to the best solution. Let b denote the best score for an instance, then the range $[0, b]$ is scaled to $[0, 1]$ for MAX-AREA and the range $[b, 1]$ is scaled to $[0, 1]$ for MIN-AREA. This removes the influence of the bound quality from the score. We see a significant difference for MIN-AREA but only a small difference for MAX-AREA.

11.5. Conclusions

The 2019 CG Challenge motivated a considerable number of teams to engage in intensive optimization studies. This success has not only led to practical progress on the problem of area optimization, but also turned the CG Challenge into a continuing feature of CG Week, spawning considerable work through the 2020 Challenge problem (Minimum Convex Partition) and the 2021 Challenge problem (Coordinated Motion Planning). This promises to motivate further work on the involved problems, as well as other practical geometric optimization work. We are confident that this will further strengthen the bridges between optimization theory and practical algorithm engineering.

This chapter gives an overview of the 2020 Computational Geometry Challenge, which targeted the problem of partitioning the convex hull of a given planar point set P into the smallest number of convex faces, such that no point of P is contained in the interior of a face.

12.1 Introduction	255
12.2 Related Work	255
12.3 Instances	256
12.4 Evaluation	257
12.5 Categories	257
12.6 Server and Timeline . . .	258
12.7 Outcomes	258
12.8 Conclusions	259

12.1. Introduction

The 2019 CG Challenge focused on the problem of computing minimum-area polygons whose vertices were a given set of points in the plane. This Challenge generated a strong response from many research groups, from both the Computational Geometry and the combinatorial optimization communities, and resulted in a lively exchange of solution ideas.

For CG Week 2020, the second CG:SHOP Challenge became an event within the CG Week program, with top performing solutions reported in the Symposium on Computational Geometry proceedings. The schedule for the Challenge was advanced earlier, to give an opportunity for more participation, particularly among students, e.g., as part of course projects.

The specific problem that formed the basis of the 2020 CG Challenge was the following:

Problem 12.1.1 (MINIMUM CONVEX PARTITION (MCP) in the plane.)

Given: A set P of n points in the plane.

Goal: A plane graph with vertex set P (with each point in P having positive degree) that partitions the convex hull of P into the smallest possible number $u(P)$ of convex faces.

Note that collinear points are allowed on face boundaries. Each internal face angle at each point of P is at most π .

12.2. Related Work

The problem of computing a partition of a simple polygon, having n vertices, into a minimum number of convex pieces has been well studied. If Steiner points are allowed to be added, then Chazelle and Dobkin [302] gave an algorithm, based on dynamic programming, which computes exactly an optimal solution in time $O(n + r^3)$, where r is the number of reflex vertices of the given polygon. For decompositions using only diagonals (no Steiner points), Greene [303] gave an $O(r^2 n^2)$ algorithm, also based on dynamic programming. Keil [304] improved the running

* A preliminary version [12] of this chapter was published on arXiv. Many thank to the co-organizers Erik Demaine, Sándor Fekete, Phillip Keldenich, and Joseph Mitchell for all their contributions.

time to $O(rn^2 \log n)$ and gave a proof of NP-hardness for polygons with holes; later, Keil and Snoeyink [305] improved the complexity to $O(n + r^2 \min\{r^2, n\})$.

The complexity of MINIMUM CONVEX PARTITION (MCP) in the plane was unknown when the 2020 CG Challenge began in September 2019. In November 2019, Grelier announced a proof of NP-hardness for the case of planar point sets in not necessarily general position [306]. The complexity of the MCP for points in general position is still open at the time of this writing. On the positive side, a number of positive algorithmic results have been known for a while, assuming special properties of P . For point sets that can be decomposed into a limited number of convex layers, Fevens, Meijer and Rappaport [307] gave a polynomial-time algorithm. Assuming that no three points are collinear, Knauer and Spillner [308] gave a 3-approximation algorithm that runs in $O(n \log n)$, and a $\frac{30}{11}$ -approximation of complexity $O(n^2)$.

Worst-case bounds have also been considered; for this purpose, define $U(n)$ as the maximum number $u(P)$ over all non-degenerate point sets P with $n \geq 3$. In 1998, Urrutia [309] conjectured that $U(n) \leq n+1$. Neumann-Lara, Rivera-Campo and Urrutia [310] showed that $U(n) \leq \frac{10n-18}{7}$; Lomeli-Haro [311] showed that $U(n) \leq \frac{10}{7}n - h$, where h is the number of points on the convex hull. This bound was improved by Hosono [312] to $U(n) \leq \frac{7(n-3)}{5}$, and later by Sakai and Urrutia [313] to $U(n) \leq \frac{4}{3}n - 2$. Conversely, Knauer and Spillner [308] showed that $U(n) \geq n + 2$, which was improved by García-López and Nicolás [314] to $U(n) \geq \frac{12}{11}n - 2$.

Aiming for solutions to benchmark instances, Barboza, Souza and Rezende [315] gave an integer programming formulation of the MCP, and showed that this can be used to solve instances with up to 50 points to provable optimality. Recently, after the challenge, Cambazard and Catusse [316] proposed an improved formulation that can solve instances with up to 100 points to provable optimality and provide lower bounds for instances with up to 300 points.

12.3. Instances

The contest started with a total of 247 benchmark instances, as follows. Each of these instance consisted of n points in the plane with integer coordinates. For $n \in \{10, 15, 20, 25, 30, 35, 40, 45, 50, 60, 70, 80, 90, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10\,000, 20\,000, 30\,000, 40\,000, 50\,000, 60\,000, 70\,000, 80\,000, 90\,000, 100\,000\}$, there were six instances each. In addition, there is one instance of size $n = 1\,000\,000$.

The instances were of four different types:

- ▶ **uniform:** uniformly at random from a square.
- ▶ **edge:** randomly generated according to the distribution of the rate of change (the “edges”) of an image.
- ▶ **illumination:** randomly generated according to the distribution of brightness of an image (such as an illumination map).

- **orthogonally collinear points:** randomly generated on an integral grid to have a lot of collinear points (similar to PCBs and distorted blueprints).

These instances were based on point sets that were originally generated for the 2019 Challenge; as such, they tended to be in general position. To account for this and the progress in complexity (which was based on instances with collinear points), a further 99 instances with larger numbers of collinear points were added in January 2020.

12.4. Evaluation

The comparison between different teams was based on an overall *score*. For each instance, this score is a number between 0 and 1, with higher scores corresponding to better solutions. The trivial solution, i.e., a triangulation, corresponds to a score of 0, and a solution without any edges, which is, of course, infeasible, corresponds to a score of 1.

For an instance, i.e., a point set P consisting of n points, let c be the number of points on the convex hull of P . Observe that any triangulation of P is a convex partition with $2n - 2 - c$ bounded faces and $3(n - 1) - c$ edges. Moreover, any convex partition Π can be obtained by starting with a triangulation containing its edges, and removing the excess edges one by one. In this process, removing a single edge also decreases the number of bounded faces by exactly 1. Thus, any solution Π with $f = 2n - 2 - c - s$ faces for some $s \geq 0$ has $m = 3(n - 1) - c - s$ edges and vice versa. This allows using the number $m(\Pi)$ of edges in a solution Π instead of the number of faces to determine the score of Π as

$$\text{score}(\Pi) := \frac{s(\Pi)}{3(n - 1) - c},$$

where $s(\Pi) = 3(n - 1) - c - m(\Pi)$. In other words, the score for a solution to an instance P is the fraction of edges removed from a triangulation of P .

The total score achieved by each team was the sum of all individual instance scores; only the best feasible solution submitted was used to compute the score. Participation required submitting feasible solutions. Feasibility was checked at the time of upload. Failing to submit a feasible solution for an instance resulted in a default score of 0 for that instance.

In case of ties, the tiebreaker was set to be the time a specific score was obtained. This turned out not to be necessary.

12.5. Categories

The contest was run in an *Open Class*, in which participants could use any computing device, any amount of computing time (within the duration of the contest) and any team composition. In the *Junior Class*, a team was required to consist exclusively of participants who were eligible according to the rules of CG:YRF (the *Young Researchers Forum* of CG Week), defined as not having defended a formal doctorate before 2018.

The demand for an additional *Limited Class* (to be run on a specific server that was to be uniform for all participants) turned out to be too low to justify the additional effort.

12.6. Server and Timeline

The contest itself was run through a dedicated server at TU Braunschweig, hosted at <https://cgshop.ibr.cs.tu-bs.de/competition/cg-shop-2020/>. It opened at 18:00 CEDT (noon, EDT) on September 30, 2019, and closed at 24:00 (midnight, AoE), February 14, 2020.

12.7. Outcomes

A total of 21 teams participated in the contest. In the end, the top 10 in the leaderboard looked as shown in Table 12.1; note that according to the scoring function, a higher score is better.

The progress over time of each team's score can be seen in Figure 12.1.

Clearly, the outcome was quite tight between the top teams; in particular, Team UBC and Omega were only separated by 0.02% in their respective scores. As can be seen from Figure 12.2, OMEGA found very slightly better solutions for a large number of instances (which is also reflected by the high number of unique best solutions in Table 12.1), while Team UBC found significantly better solution for six of the instances. The latter was sufficient for the overall win. Whether the later added orthogonal instances did change the outcome is questionable, as can be seen in Figure 12.3. While the new instances did allow more optimization and larger convex areas, nearly all teams were able to do so and the solution variance in these instances is actually lower than for the other instances.

The top 3 finishers in the Open Class were invited for contributions in the 2020 SoCG proceedings, as follows.

1. Team UBC: Da Wei Zheng, Jack Spalding-Jamieson and Brandon Zhang [317].

Table 12.1: The top of the final leaderboard. "Best solutions" are the best found by any participating team, which does not exclude the possibility of better solutions. "Unique best" solutions are those that were not found by any other team.

Position	Team	Score	# best solutions	# unique best solutions
1	Team UBC	175.172880	209	11
2	OMEGA	175.130597	297	126
3	CGA-Sbg	175.040207	187	0
4	Les Shadoks	174.695586	160	6
5	G-SCOP	174.543068	138	0
6	Min2Win@Zurich	174.384784	121	0
7	TUFUnky4you	173.973716	100	0
8	Team Technion	173.621857	99	0
9	Sapucaia, de Rezende, de Souza	170.939574	88	0
10	ucsbtheorylab	169.975180	76	0

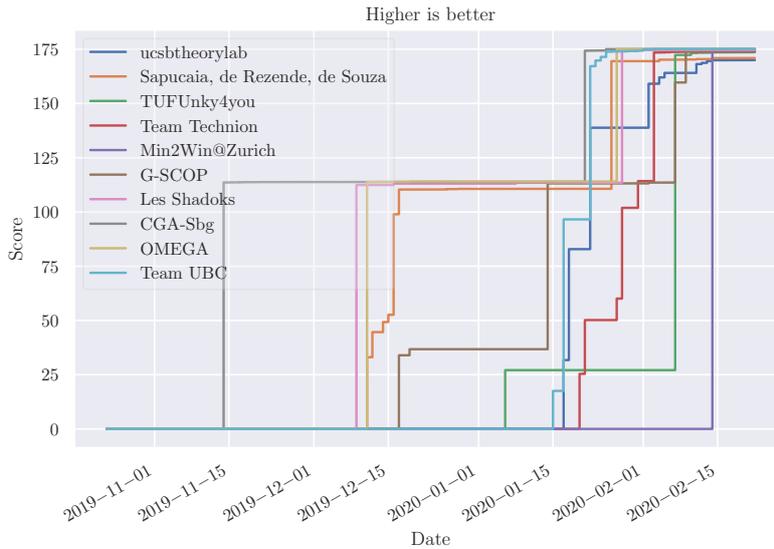


Figure 12.1: Total score over time for the best ten teams. The jump in the last quarter is partially due to the extended instance set.

2. Team OMEGA: Laurent Moalic, Dominique Schmitt, Julien Lepagnot and Julien Kritter [318].
3. Team CGA-Sbg: Günther Eder, Martin Held, Stefan de Lorenzo and Peter Palfrader [319].

Consisting only of students, Team UBC was also the runaway winner of the Junior Class.

All three teams engineered their solutions based, broadly, on variants of local search methods, with the use of randomization and constraint programming [317], genetic approaches [318], and tailored initial decompositions [319]. Details of their methods and the engineering decisions they made are given in their respective papers.

12.8. Conclusions

The 2020 CG:SHOP Challenge motivated a considerable number of teams to engage in intensive optimization studies. Not only did this lead to practical developments, it also triggered theoretical progress, as demonstrated by Grelier [306]. We are confident that this will motivate further work on the problem of Minimum Convex Partitions, as well as other practical geometric optimization work.

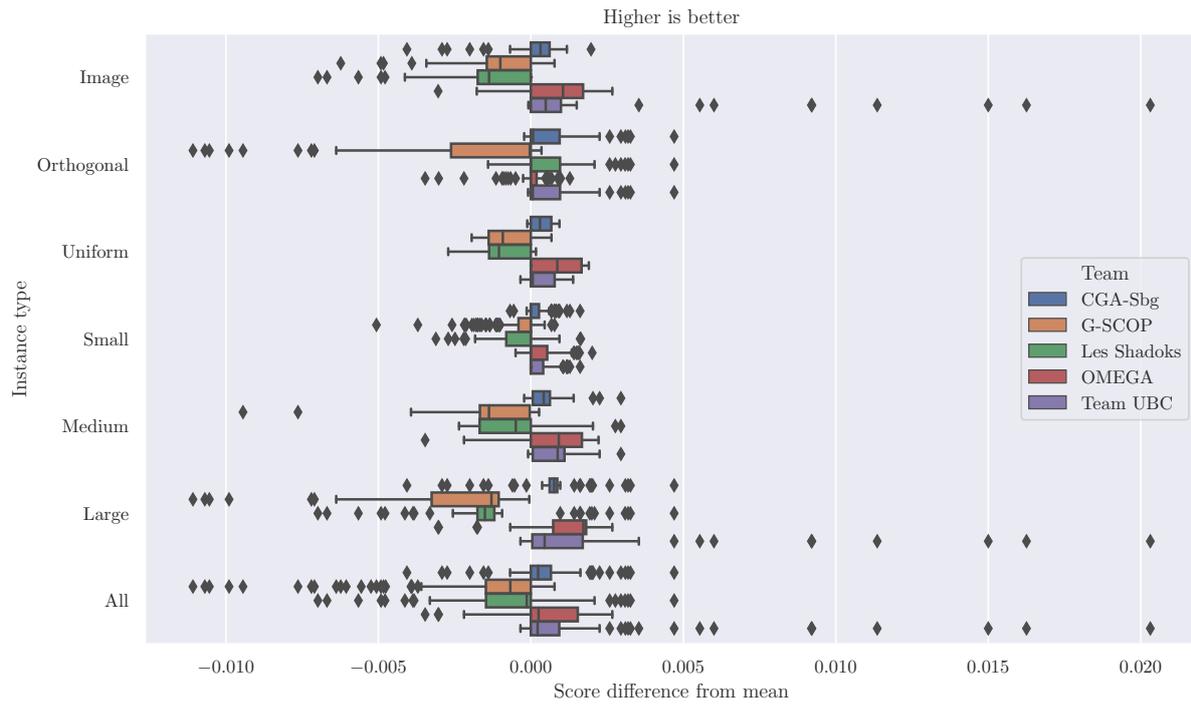


Figure 12.2.: Differences in the scores for individual instances for the top teams compared to their mean. We can see that Team OMEGA is best in more instances but only by reasonably small margins while Team UBC is significantly better in a few large image-based instances that make the difference.

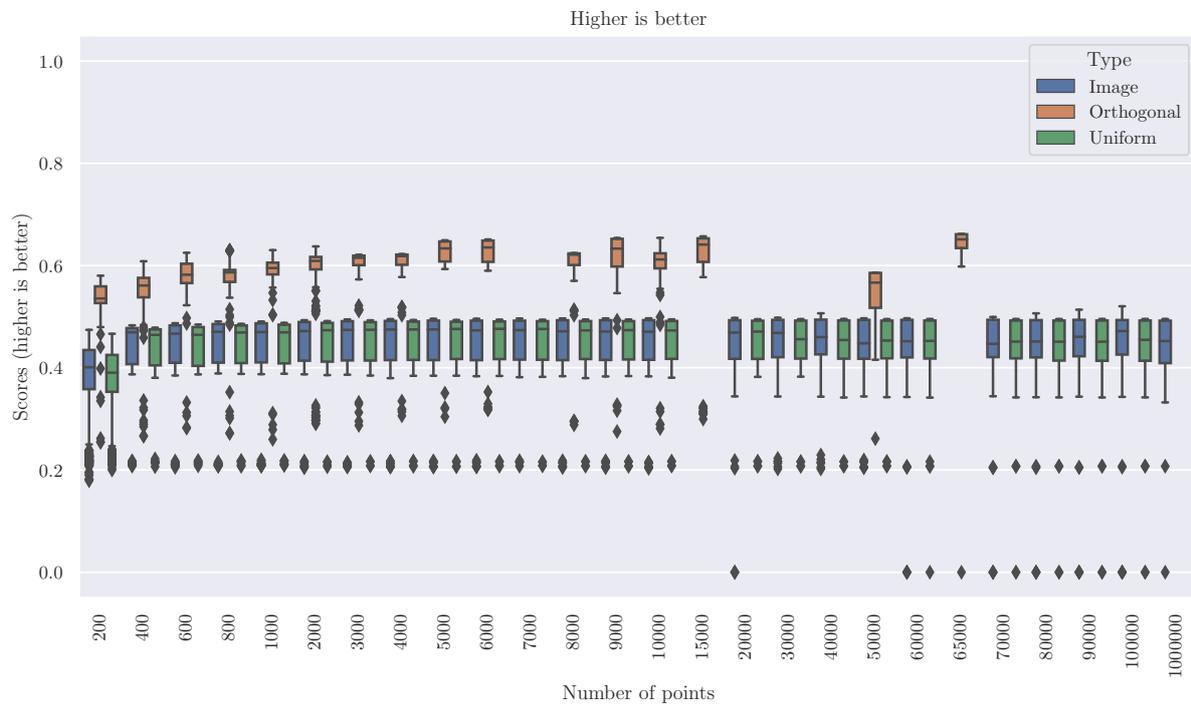


Figure 12.3.: Distribution of the best instance scores of the teams over size. Only scores for submitted instances are considered such that larger instances are based on slightly less points. A lower variance indicates that all teams had very similar solutions and their approaches performed similar. A large variance indicates that the approaches of the teams performed very differently. The image-based and uniform instances are barely distinguishable in the score distribution. Thus, the approaches of the teams probably performed similar well on both types. The orthogonal instances on the other hand look very different. Surprisingly, the variance is much lower while allowing much more optimization as visible by the higher mean score. Based on this, it is questionable whether adding the orthogonal instances did really influence the competition much.

This chapter gives an overview of the 2021 Computational Geometry Challenge, which targeted the problem of optimally coordinating a set of robots by computing a family of collision-free trajectories for a set S of n pixel-shaped objects from a given start configuration to a desired target configuration.

13.1. Introduction

For CG Week 2020, the second CG:SHOP Challenge became an event within the CG Week program, with top performing solutions reported in the Symposium on Computational Geometry proceedings. The schedule for the Challenge was advanced earlier, to give an opportunity for more participation, particularly among students, e.g., as part of course projects.

The third edition of the Challenge in 2021 continued the 2020 format, leading to contributions in the SoCG proceedings.

13.2. The Challenge: Coordinated Motion Planning

Coordinating the motion of a set of objects is a fundamental problem that occurs in a large spectrum of theoretical contexts and practical applications. A typical task arises from relocating a large collection of agents from a given start into a desired goal configuration in an efficient manner, while avoiding collisions between objects or with obstacles.

13.2.1. The Problem

The specific problem that formed the basis of the 2021 CG Challenge was the following; see Figure 13.1 for a simple example.

Problem 13.2.1 (COORDINATED MOTION PLANNING of unit squares.)

Given: A set of n axis-aligned unit-square robots in the plane, a set $S = \{s_1, \dots, s_n\}$ of n distinct start pixels (unit squares) of the integer grid, and a set $T = \{t_1, \dots, t_n\}$ of n distinct target pixels of the integer grid. In addition, there may be a set of obstacles, consisting of a number of stationary, blocked pixels that cannot be used by robots at any time.

Goal: The task is to compute a feasible set of trajectories for all n robots, with the trajectory for robot i moving it from s_i to t_i , such that the overall schedule

13.1	Introduction	261
13.2	The Challenge	261
13.2.1	The Problem	261
13.2.2	Related Work	262
13.2.3	Instances	264
13.2.4	Evaluation	266
13.2.5	Categories	266
13.2.6	Server and Timeline	266
13.3	Outcomes	266
13.4	Conclusions	268

* A preliminary version [13] of this chapter was published on arXiv. Many thanks to the co-organizers Sándor Fekete, Phillip Keldenich, and Joseph Mitchell for all their contributions.

is optimal with respect to an objective function.

In order to be feasible, a trajectory must satisfy a number of conditions. During each unit of time, each robot can move (at unit speed) in a direction (north, south, east or west) to an adjacent pixel, provided the robot remains disjoint from all other robots during the motions. This condition has to be satisfied at all times, not just when robots are at pixel positions. For example, if there are robots at each of the two adjacent pixels (x, y) and $(x + 1, y)$, then the robot at (x, y) can move east into position $(x + 1, y)$ only if the robot at $(x + 1, y)$ moves east at the same time, so that the two robots remain in contact, during the movement, but never overlap.

The contest was run on two different objective functions, as follows.

MAX: Minimize the makespan, i.e., the time until all robots have reached their targets.

SUM: Minimize the total sum of distances traveled by all robots.

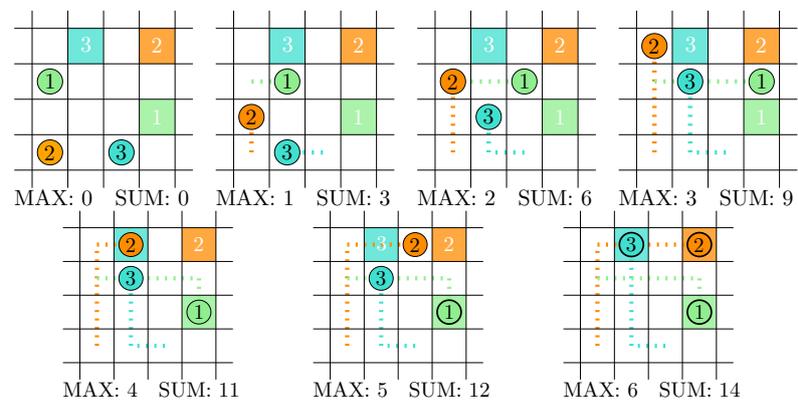


Figure 13.1: A feasible sequence of positions that minimizes both makespan and total distance. Robot positions are shown by circles, target positions by squares.

13.2.2. Related Work

Coordinating the motion of many agents plays a central role when dealing with large numbers of moving robots, vehicles, aircraft, or people. How can each agent choose an efficient route that avoids collisions with other agents as they simultaneously move to their targets? These basic questions arise in many applications, such as ground swarm robotics [320, 321], aerial swarm robotics [322, 323], air traffic control [324], and vehicular traffic networks [325, 326].

Multi-robot coordination dates back to the early days of robotics and Computational Geometry. The seminal work by Schwartz and Sharir [327] from the 1980s considers coordinating the motion of disk-shaped objects among obstacles. Their algorithms are polynomial in the complexity of the obstacles, but exponential in the number of disks. Hopcroft et al. [328] and Hopcroft and Wilfong [329] proved PSPACE-completeness of moving multiple robots to a target configuration, showing the significant challenge of coordinating many robots.

There is a vast body of other related work dealing with multi-robot motion planning, both from theory and practice. For a more extensive overview, see [330]. In both discrete and geometric variants of the problem, the

objects can be *labeled*, *colored*, or *unlabeled*. In the *labeled* case, the objects are all distinguishable and each object has its own, uniquely defined target position. In the *colored* case, the objects are partitioned into k groups and each target position can only be covered by an object with the right color. This was considered by Solovey and Halperin [331], who present and evaluate a practical sampling-based algorithm. In the *unlabeled* case, objects are indistinguishable and target positions can be covered by any object.

This scenario was first considered by Kloder and Hutchinson [332], who presented a practical sampling-based algorithm. Turpin et al. [333] give an algorithm for finding a solution in polynomial time, if one exists. This is optimal with respect to the longest distance traveled by any one robot, but only holds for disk-shaped robots under additional restrictive assumptions on the free space. For unit disks and simple polygons, Adler et al. [334] provide a polynomial-time algorithm under the additional assumption that the start and target positions have some minimal distance from each other. Under similar separability assumptions, Solovey et al. [335] provide a polynomial-time algorithm that produces a set of paths that is no longer than $\text{OPT} + 4m$, where m is the number of robots and OPT is the total length of the paths in an optimal solution. However, they do not consider the makespan, but only the total path length. On the negative side, Solovey and Halperin [336] prove that the unlabeled multiple-object motion planning problem is PSPACE-hard, even when restricted to unit square objects in a polygonal environment.

There is also a wide range of practical related work. Self-configuration of robots as active agents was studied by Naz et al. [337]. A basic model in which robots are used as building material was introduced by Derakhshandeh et al. [221, 338]. This resembles Claytronics robots like Catoms; see Goldstein and Mowry [339]. In more recent work, Thalamy et al. [340] consider using scaffolding structures for asynchronous reconfiguration.

For an instance of parallel reconfiguration, a lower bound for the time required for *all* robots to reach their targets is the time it takes to move just *one* robot to its target in the absence of other robots, i.e., by the maximum distance between a robot's origin and target. Moving a dense arrangement of robots to their targets while avoiding collisions may require substantially more time than this lower bound. This motivates the *stretch factor*, which is defined to be the ratio of the time taken by a parallel motion plan divided by the simple lower bound.

In recent work, Demaine et al. [223, 330] provide several fundamental insights into these problems of coordinated motion planning for the scenario with labeled robots. They develop algorithms that (under relatively mild assumptions on the separation between robots, slightly more generous than provided in the Challenge) achieve *constant* stretch factors that are independent of the number of robots. Thus, these algorithms provide an absolute performance guarantee on the makespan of the parallel motion schedule, which implies that the schedule is a constant-factor approximation of the best possible schedule. For densely packed arrangements of robots (without separation assumptions), they proved that a constant stretch factor is no longer possible, and gave upper and

lower bounds on the worst-case stretch factor. See Figure 13.2 for an illustration.

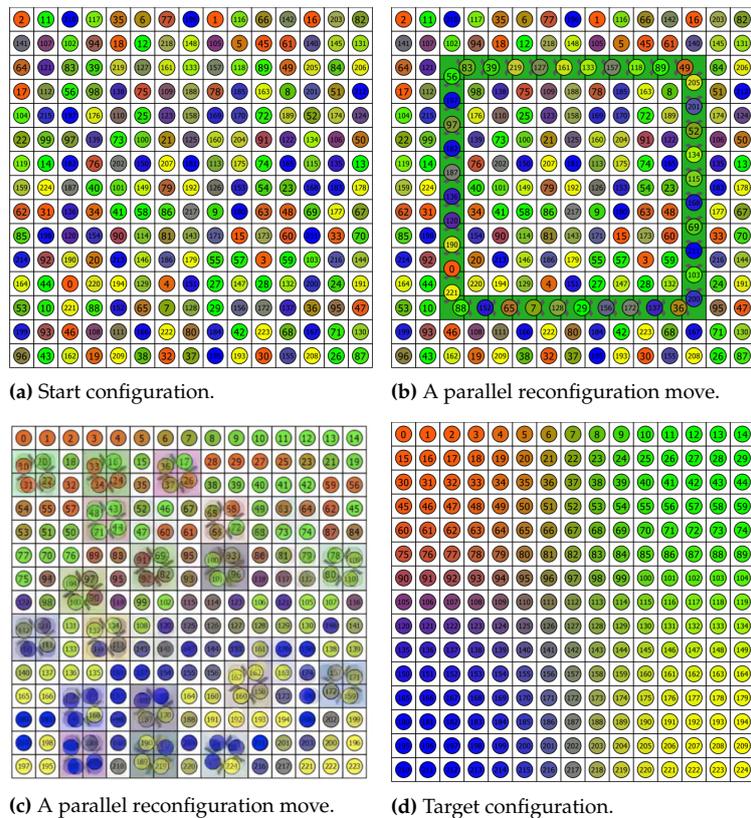


Figure 13.2: Parallel reconfiguration, established by [223, 330]: See <https://www.ibr.cs.tu-bs.de/users/fekete/Videos/CoordinatedMotionPlanning.mp4> for a video [224].

13.2.3. Instances

An important part of any challenge is the creation of suitable instances. If the instances are easy to solve to optimality, the challenge becomes trivial. If instances require a huge amount of computation to find any decent solutions, the challenge may heavily favor teams that can afford better computation equipment. The same is true if the set of instances becomes too large to manage with a single (or few) computers.

A priori, it is difficult to tell how hard finding a good solution to an instance is and which parameters influence the difficulty of solving an instance (and in what way). Therefore, it is important to create a set of instances that are diverse with respect to parameters that are likely to influence their difficulty.

To create interesting and challenging instances, we thus developed an instance generator that can be tuned by adjusting several parameters to create diverse instances. Basic parameters control the size of the map, the density to which the map is filled with robots, and the distribution functions for start and target positions. For the distribution functions, we used uniform distributions and distributions based on various images, such as microscope images of bacteria colonies. The image-based instances often have interesting, natural patterns; their accumulations of robots in some areas were expected to yield challenging instances. Additionally,

we created artificial bottlenecks and accumulations of robots by inserting obstacles and clusters of robots to increase the difficulty.

Obstacles are created by randomly placing rectangles with (truncated) normally distributed sizes. Any area that is completely surrounded by obstacles (i.e., any hole in the union of obstacle rectangles) also becomes an obstacle to ensure that instances remain feasible. Despite its simplicity, this procedure is able to create various interesting obstacles after some parameter tuning; see Figure 13.3 for an example.

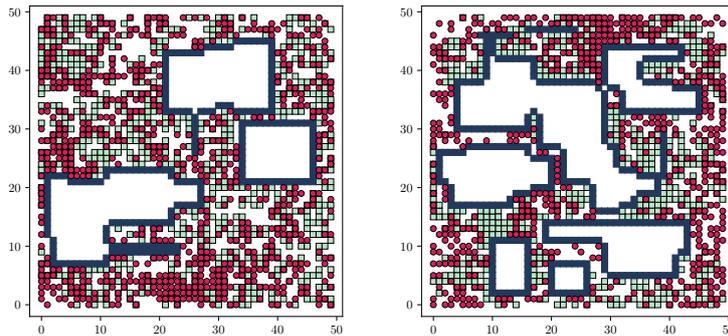


Figure 13.3.: A visualization of two instances with obstacles (dark pixels). Start and target positions of robots are indicated by red disks and green squares.

The idea of robot clusters is to have groups of robots that are close to each other in the start and target configurations, but have to travel a significant distance as a group. During its motion, such a robot cluster may have to pass through narrow obstacles or may clash with other clusters, yielding interesting conflicts. Robot clusters are created by specifying the desired number of clusters and the parameters of a normal distribution controlling their size. For each cluster, a start and a target is randomly selected according to the distribution functions controlling the robots' start and target positions. A corresponding number of robots is then distributed within size-dependent windows around the start and target positions of the cluster. These windows may be too small to contain the start and target positions due to obstacles or robots that have already been placed; in that case, we retry with a larger window.

As described above, certain parameters (such as the density) can be controlled directly by our generator. To ensure that our instances are diverse even with respect to parameters for which this is not possible, we generated a large set of instances by selecting a diverse set of possible values for each parameter of our generator and then generating several instances for each possible combination of these parameter values.

This resulted in a set of more than 10 000 candidate instances from which we then selected the challenge instances as follows. We measured several properties for each of the generated instances, including the number of robots, the density, the number of robot clusters generated, the number of robots that were part of a robot cluster and the volume and free area of the map. Based on these properties, we then defined and tuned a distance function between the instances and used a greedy dispersion algorithm to select a total of 200 diverse instances from the candidates; see Figure 13.6 for an overview of the distribution of some important properties across the selected instances. See Section 10.4 for a more detailed explanation of such an procedure.

An analysis of the difficulty regarding these various parameters can be seen at the end in Figures 13.7 and 13.8.

13.2.4. Evaluation

The contest was run on a total of 203 instances, 3 of which were small, manually generated instances. For either objective function, a team's score for an instance I is the ratio L/V between L , the objective value of the best submitted solution for I , and V , the best objective value of any valid solution for I submitted by the team. The score for each instance is thus a number between 0 and 1, with 1 being the score awarded to the teams with the best submitted solution and 0 being a default score. As a consequence, the total score is a number between 0 and 203 for either contest, where 203 is best possible. For each contest (MAX and SUM), participants were compared based on their total scores; the winner for each contest was the one with the highest score.

In case of ties, the tiebreaker was set to be the time a specific score was obtained. This turned out not to be necessary.

13.2.5. Categories

The contest was run in an *Open Class*, in which participants could use any computing device, any amount of computing time (within the duration of the contest) and any team composition. In the *Junior Class*, a team was required to consist exclusively of participants who were eligible according to the rules of CG:YRF (the *Young Researchers Forum* of CG Week), defined as not having defended a formal doctorate before 2019.

13.2.6. Server and Timeline

The contest itself was run through a dedicated server at TU Braunschweig, hosted at <https://cgshop.ibr.cs.tu-bs.de/competition/cg-shop-2021/>. It opened at 00:00 AoE on November 20, 2020, and closed at 24:00 (midnight, AoE), February 15, 2021.

13.3. Outcomes

A total of 17 teams submitted solutions. In the end, the leaderboard for the top 10 teams in both categories looked as shown in Table 13.1 and Table 13.2; note that according to the scoring function, a higher score is better.

The progress over time of each team's score can be seen in Figure 13.5; the best solutions for all instances (displayed by score) can be seen in Figure 13.4.

There were three clear frontrunners, with Team Shadoks placing first and third in MAX and SUM, respectively; Team UNIST came in second in both categories, while Team gitastrophe placed third and first in MAX and SUM. A closer look at the scores reveals that Team Shadoks achieved

Position	Team	Score	# best solutions
1	Shadoks	202.9375	202
2	UNIST	174.0180	14
3	gitastrophe ^J	159.5472	24
4	S10ppy J035 ^J	109.2778	3
5	École Polytechnique ^J	90.8213	3
6	TUeSWarM ^J	83.0897	7
7	JoJo ^J	75.2449	6
8	BlueTeamTechnion ^J	65.0906	7
9	Lasteam ^J	41.6065	1
10	Kleinkariert ^J	36.0898	0

Table 13.1: The top of the final leaderboard for MAX. “Best solutions” are the best found by any participating team, which does not exclude the possibility of better solutions, but provides a score of 1 for that instance; scores are truncated to four decimal places. Junior teams are indicated by ^J.

Position	Team	Score	# best solutions
1	gitastrophe ^J	198.4943	57
2	UNIST	191.7893	120
3	Shadoks	180.4952	0
4	cgi@tau	175.0754	6
5	BlueTeamTechnion ^J	165.5633	34
6	TUeSWarM ^J	146.2244	1
7	S10ppy J035 ^J	133.450	1
8	Team ITI ^J	109.4631	0
9	Kleinkariert ^J	81.7086	0
10	École Polytechnique ^J	75.4734	0

Table 13.2: The top of the final leaderboard for SUM. “Best solutions” are the best found by any participating team, which does not exclude the possibility of better solutions, but provides a score of 1 for that instance; scores are truncated to four decimal places. Junior teams are indicated by ^J.

an almost perfect score overall in MAX, which was sufficient to place first in the sum of both objective functions, at 401.4318. Team UNIST managed a balanced outcome for both objective functions, for a combined score of 365.8073. Conversely, Team gitastrophe achieved the best score for SUM, but a slightly inferior result for MAX, resulting in a combined score of 358.0415; at the same time, they also placed first in all three events in the Junior Class.

These top 3 finishers were invited for contributions in the 2021 SoCG proceedings, as follows.

1. Team Shadoks: Loïc Crombez, Guilherme D. da Fonseca, Yan Gerard, Aldo Gonzalez-Lorenzo, Pascal Lafourcade, Luc Libralleso [341].
2. Team UNIST: Hyeyun Yang, Antoine Vigneron [342].
3. Team gitastrophe: Jack Spalding-Jamieson, Paul Liu, Brandon Zhang, Da Wei Zheng [343].

All three teams engineered their solutions based on a spectrum of heuristics for generating start configurations, in combination of a variety of local search methods, including simulated annealing, k -optimization, and more refined, tailor-made approaches. Details of their methods and the engineering decisions they made are given in their respective papers.

13.4. Conclusions

The 2021 CG:SHOP Challenge motivated a considerable number of teams to engage in intensive optimization studies. The outcomes promise further insight into the underlying, important optimization problem. Moreover, the considerable participation of junior teams indicates that the Challenge itself motivates a considerable number students and young researchers to work on practical algorithmic problems.

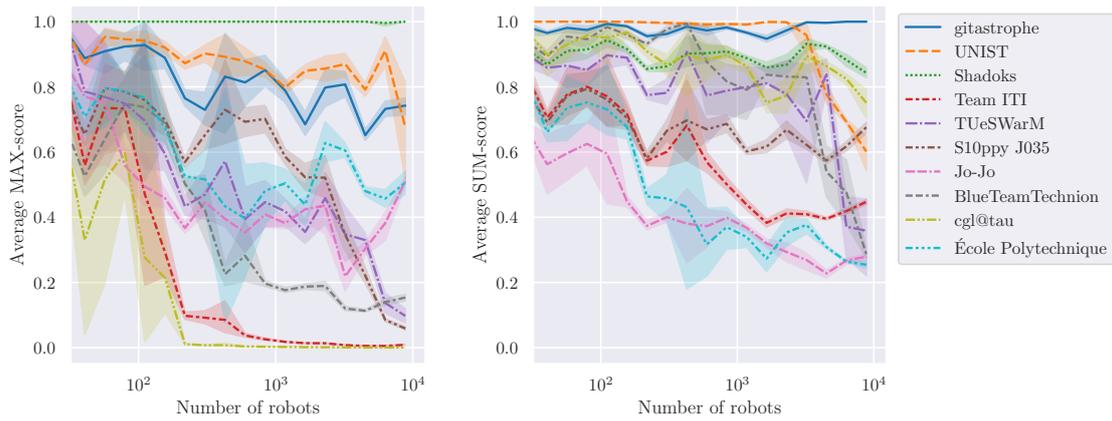
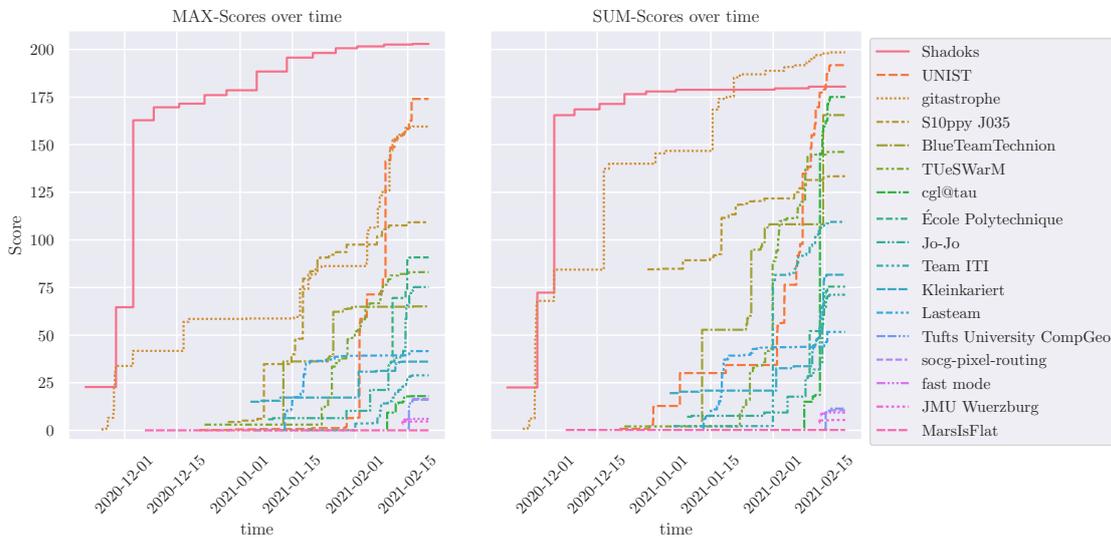


Figure 13.4.: The scores over the instance sizes of the ten best teams. We can see that team Shadoks is nearly unbeaten in the MAX objective. Team UNIST is very strong for smaller instances and the SUM objective, but the approach of team gitastrophe seems to scale better and wins on the larger instances by a significant margin.



(a) MAX

(b) SUM

Figure 13.5.: Progress of the teams over time. Team Shadoks built its lead in the MAX-objective very early and only did minor improvements over the rest of the time. For team gitastrophe, a continuous progress is visible. Team UNIST, on the other hand, started relatively late. Most teams seem to have only focused on the SUM-objective, as these scores show the largest jumps, with submissions picking up in the last month of the competition.

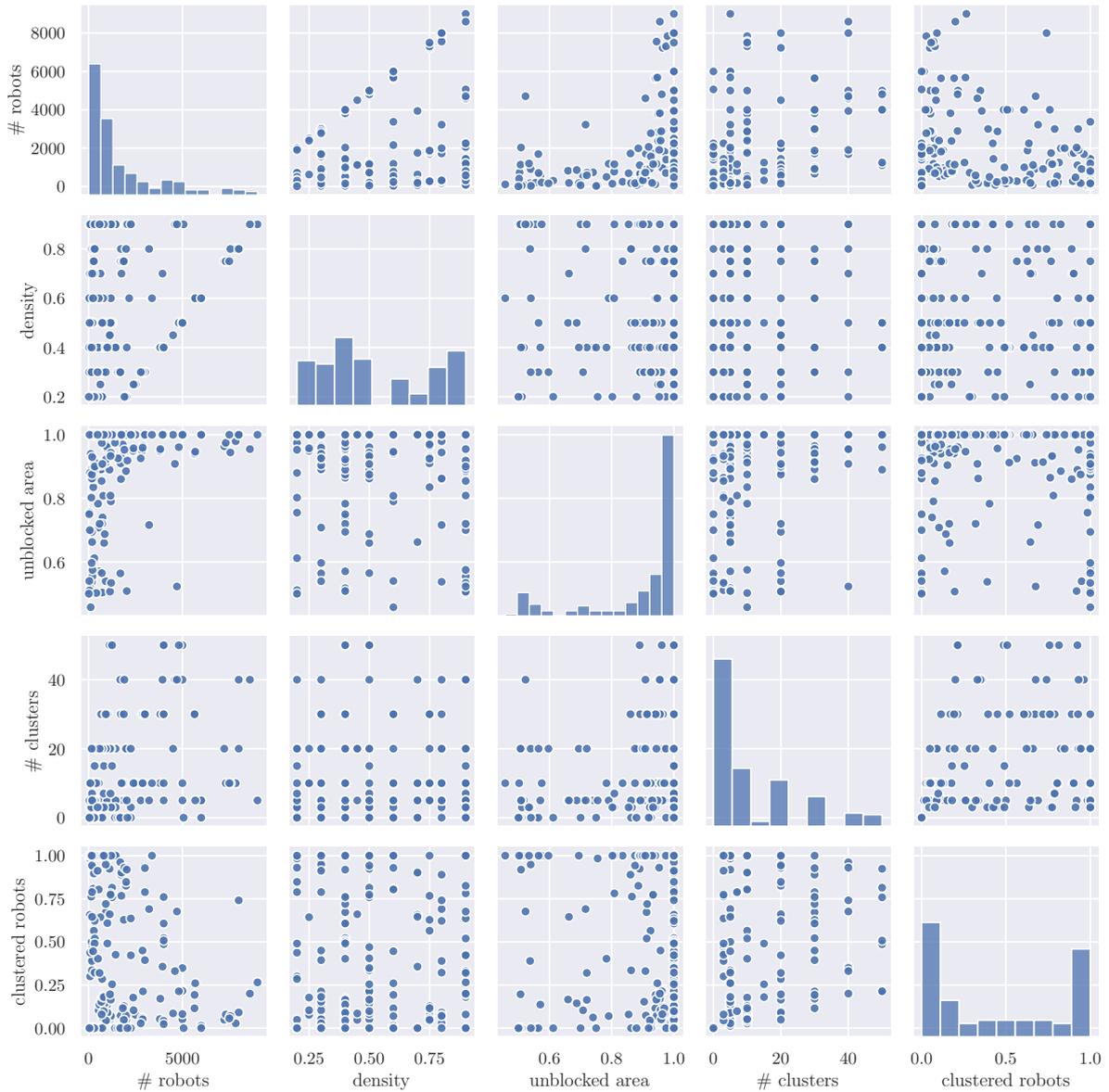


Figure 13.6.: A pair plot of the distribution of our instances according to several important instance properties. *# robots* describes the number of robots in the instance, *density* describes the ratio of occupied positions in the environment (excluding obstacles), *unblocked area* describes the ratio of reachable positions within the bounding box, *# clusters* describes how many groups of robots with similar start and target locations have been placed by the generator, and *clustered robots* describes the ratio of robots that belong to a cluster.

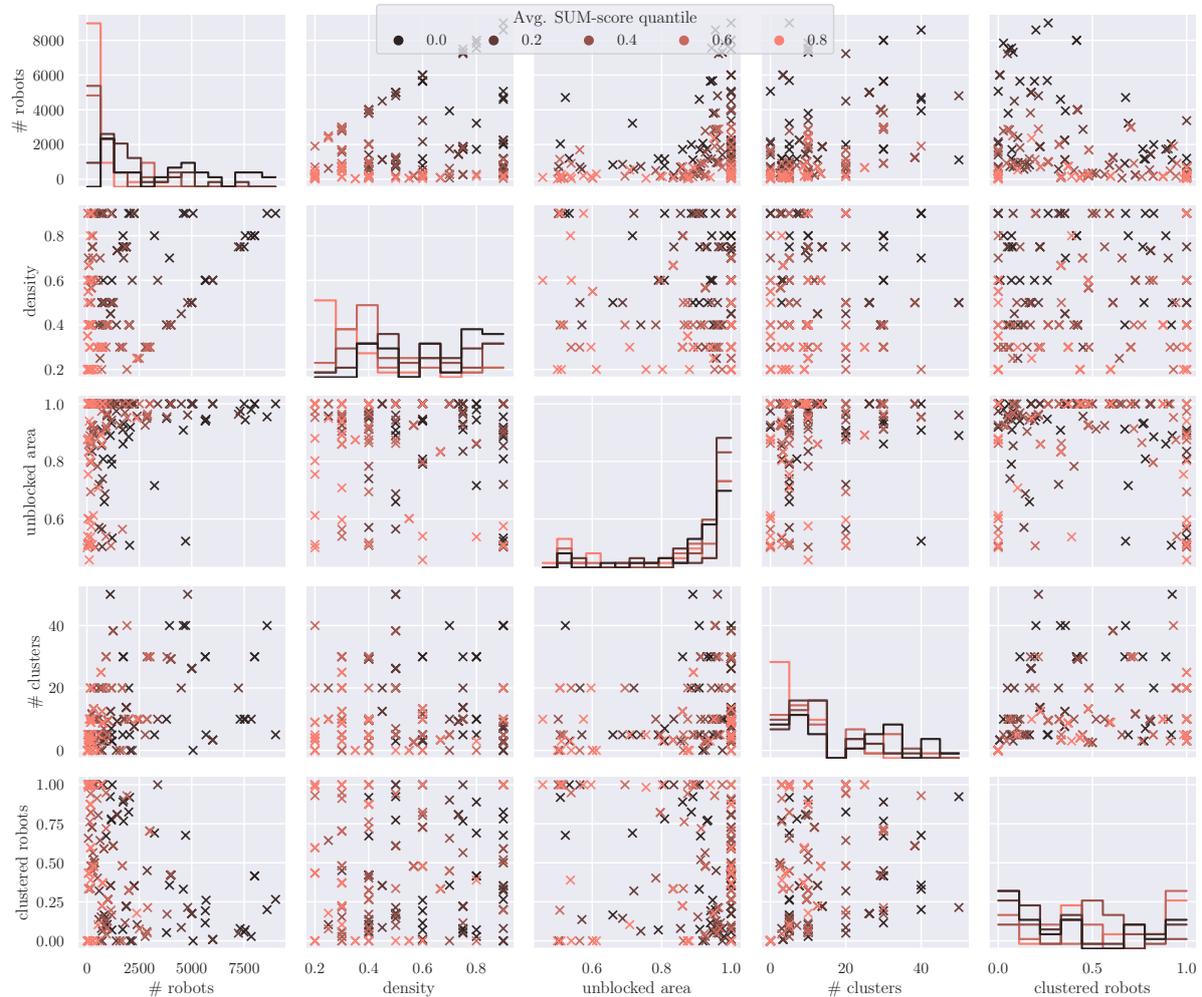


Figure 13.7.: This plot provides some clues for the practical difficulty of instances, indicated by low average scores. Because the scores are based on the best submitted solution, this implies that some teams with special approaches were able to obtain significantly better results than the “basic” approach. In the plots, we split the instances into five quantiles, such that the instances with high average score are shown in red, while darker colors indicate lower average score, i.e., instances considered to be difficult. The most dominant factor appears to be the number of robots. Because it correlates with many other parameters and the distribution is not fully homogeneous, considering individual parameters would be highly skewed. We therefore also show combinations of parameters, especially with the number of robots to counteract the effect that some parameter ranges have on average significantly fewer or more robots. The diagonals show the distribution of individual parameters; we can see that the easiest instances appear to be the small ones, because the better quantiles are nearly full. Scatter plots show combinations of two parameters, with every instance shown as one point in the plot. We can also see that increasing the density without changing the number of robots makes the instances more difficult. Moreover, instances without obstacles seem to be easier to solve. Due to lack of data, not much can be said for clusters, with difficulty mostly correlating with the number of robots in the available data points.

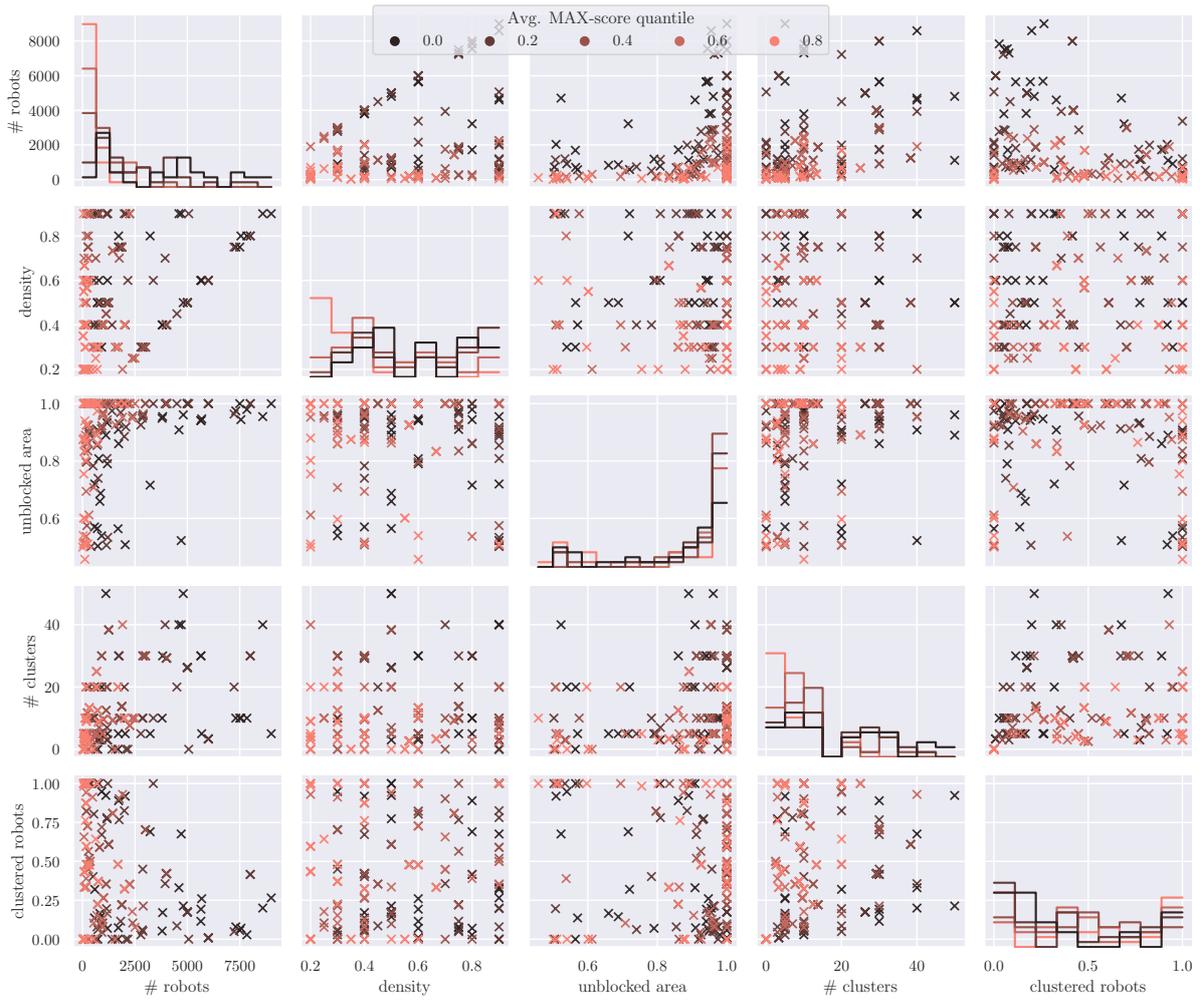


Figure 13.8.: A detailed analysis for MAX instead of SUM; refer to Figure 13.7 for the breakdown into subfigures.

Bibliography

- [1] Sándor P. Fekete, Linda Kleist, and Dominik Krupke. ‘Minimum Scan Cover with Angular Transition Costs’. In: *36th International Symposium on Computational Geometry, SoCG 2020, June 23-26, 2020, Zürich, Switzerland*. Ed. by Sergio Cabello and Danny Z. Chen. Vol. 164. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 43:1–43:18. doi: [10.4230/LIPIcs.SoCG.2020.43](https://doi.org/10.4230/LIPIcs.SoCG.2020.43) (cited on pages ix, 9).
- [2] Sándor P. Fekete, Linda Kleist, and Dominik Krupke. ‘Minimum Scan Cover with Angular Transition Costs’. In: *SIAM J. Discret. Math.* 35.2 (2021), pp. 1337–1355. doi: [10.1137/20M1368161](https://doi.org/10.1137/20M1368161) (cited on pages ix, 9).
- [3] Kevin Buchin, Sándor P. Fekete, Alexander Hill, Linda Kleist, Irina Kostitsyna, Dominik Krupke, Roel Lambers, and Martijn Struijs. ‘Minimum Scan Cover and Variants - Theory and Experiments’. In: *19th International Symposium on Experimental Algorithms, SEA 2021, June 7-9, 2021, Nice, France*. Ed. by David Coudert and Emanuele Natale. Vol. 190. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 4:1–4:16. doi: [10.4230/LIPIcs.SEA.2021.4](https://doi.org/10.4230/LIPIcs.SEA.2021.4) (cited on pages ix, 9).
- [4] Dominik Krupke, Volker Schaus, Andreas Haas, Michael Perk, Jonas Dippel, Benjamin Grzesik, Mohamed Khalil Ben Larbi, Enrico Stoll, Tom Haylock, Harald Konstanski, Kattia Flores Pozo, Mirue Choi, Christian Schurig, and Sándor P. Fekete. ‘Automated Data Retrieval from Large-Scale Distributed Satellite Systems’. In: *15th IEEE International Conference on Automation Science and Engineering, CASE 2019, Vancouver, BC, Canada, August 22-26, 2019*. IEEE, 2019, pp. 1789–1795. doi: [10.1109/COASE.2019.8843045](https://doi.org/10.1109/COASE.2019.8843045) (cited on pages ix, 79).
- [5] Sándor P. Fekete and Dominik Krupke. ‘Practical Methods for Computing Large Covering Tours and Cycle Covers with Turn Cost’. In: *Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments, ALENEX 2019, San Diego, CA, USA, January 7-8, 2019*. Ed. by Stephen G. Kobourov and Henning Meyerhenke. SIAM, 2019, pp. 186–198. doi: [10.1137/1.9781611975499.15](https://doi.org/10.1137/1.9781611975499.15) (cited on pages ix, 12, 95, 98, 100, 105, 109, 113, 123).
- [6] Sándor P. Fekete, Alexander Hill, Dominik Krupke, Tyler Mayer, Joseph S. B. Mitchell, Ojas Parekh, and Cynthia A. Phillips. ‘Probing a Set of Trajectories to Maximize Captured Information’. In: *18th International Symposium on Experimental Algorithms, SEA 2020, June 16-18, 2020, Catania, Italy*. Ed. by Simone Faro and Domenico Cantone. Vol. 160. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 5:1–5:14. doi: [10.4230/LIPIcs.SEA.2020.5](https://doi.org/10.4230/LIPIcs.SEA.2020.5) (cited on pages ix, 153).
- [7] Aaron T. Becker, Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, Christian Rieck, Christian Scheffer, and Arne Schmidt. ‘Tilt Assembly: Algorithms for Micro-Factories that Build Objects with Uniform External Forces’. In: *28th International Symposium on Algorithms and Computation, ISAAC 2017, December 9-12, 2017, Phuket, Thailand*. Ed. by Yoshio Okamoto and Takeshi Tokuyama. Vol. 92. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 11:1–11:13. doi: [10.4230/LIPIcs.ISAAC.2017.11](https://doi.org/10.4230/LIPIcs.ISAAC.2017.11) (cited on page ix).
- [8] Aaron T. Becker, Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, Christian Rieck, Christian Scheffer, and Arne Schmidt. ‘Tilt Assembly: Algorithms for Micro-factories That Build Objects with Uniform External Forces’. In: *Algorithmica* 82.2 (2020), pp. 165–187. doi: [10.1007/s00453-018-0483-9](https://doi.org/10.1007/s00453-018-0483-9) (cited on pages ix, 177, 205).
- [9] Phillip Keldenich, Sheryl Manzoor, Li Huang, Dominik Krupke, Arne Schmidt, Sándor P. Fekete, and Aaron T. Becker. ‘On Designing 2D Discrete Workspaces to Sort or Classify Polyominoes’. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1–9 (cited on pages ix, 177).

- [10] Aaron T. Becker, Sándor P. Fekete, Li Huang, Phillip Keldenich, Linda Kleist, Dominik Krupke, Christian Rieck, and Arne Schmidt. ‘Targeted Drug Delivery: Algorithmic Methods for Collecting a Swarm of Particles with Uniform, External Forces’. In: *2020 IEEE International Conference on Robotics and Automation, ICRA 2020, Paris, France, May 31 - August 31, 2020*. IEEE, 2020, pp. 2508–2514. doi: [10.1109/ICRA40945.2020.9196551](https://doi.org/10.1109/ICRA40945.2020.9196551) (cited on pages ix, 203).
- [11] Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Joseph S. B. Mitchell. ‘Area-Optimal Simple Polygonalizations: The CG Challenge 2019’. In: *CoRR abs/2111.07304* (2021) (cited on pages x, 243).
- [12] Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Joseph S. B. Mitchell. ‘Computing Convex Partitions for Point Sets in the Plane: The CG: SHOP Challenge 2020’. In: *CoRR abs/2004.04207* (2020) (cited on pages x, 255).
- [13] Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Joseph S. B. Mitchell. ‘Computing Coordinated Motion Plans for Robot Swarms: The CG: SHOP Challenge 2021’. In: *CoRR abs/2103.15381* (2021) (cited on pages x, 261).
- [14] Judith Bennett, Dominik Krupke, Sepideh Sadegh, Jan Baumbach, Sándor P. Fekete, Tim Kacprowski, Markus List, and David B. Blumenthal. ‘Robust disease module mining via enumeration of diverse prize-collecting Steiner trees’. In: *Bioinformatics* (Jan. 2022). doi: [10.1093/bioinformatics/btab876](https://doi.org/10.1093/bioinformatics/btab876) (cited on page x).
- [15] Mohamed Ben Larbi, Kattia Pozo, Mirue Choi, Tom Haylok, Benjamin Grzesik, Andreas Haas, Dominik Krupke, Harald Konstanski, Volker Schaus, Sándor Fekete, Christian Schurig, and Enrico Stoll. ‘Towards the Automated Operations of Large Distributed Satellite Systems. Part 2: Classifications and Tools’. In: *Advances in Space Research* 67 (Sept. 2020). doi: [10.1016/j.asr.2020.08.018](https://doi.org/10.1016/j.asr.2020.08.018) (cited on pages x, 79).
- [16] Mohamed Ben Larbi, Kattia Pozo, Tom Haylok, Mirue Choi, Benjamin Grzesik, Andreas Haas, Dominik Krupke, Harald Konstanski, Volker Schaus, Sándor Fekete, Christian Schurig, and Enrico Stoll. ‘Towards the Automated Operations of Large Distributed Satellite Systems. Part 1: Review and Paradigm Shifts’. In: *Advances in Space Research* 67 (Aug. 2020). doi: [10.1016/j.asr.2020.08.009](https://doi.org/10.1016/j.asr.2020.08.009) (cited on pages x, 79).
- [17] Volker Schaus, Dominik Krupke, Mohamed Ben Larbi, Andreas Haas, Benjamin Grzesik, Jonas Radtke, Sándor Fekete, and Enrico Stoll. ‘Automated Constellation Management With Self-Regulating Data-Economic Actors’. In: *70th International Astronautical Congress (IAC)*. Oct. 2019 (cited on pages x, 79, 88).
- [18] Sándor P. Fekete and Dominik Krupke. ‘Covering Tours and Cycle Covers with Turn Costs: Hardness and Approximation’. In: *Algorithms and Complexity - 11th International Conference, CIAC 2019, Rome, Italy, May 27-29, 2019, Proceedings*. Ed. by Pinar Heggernes. Vol. 11485. Lecture Notes in Computer Science. Springer, 2019, pp. 224–236. doi: [10.1007/978-3-030-17402-6_19](https://doi.org/10.1007/978-3-030-17402-6_19) (cited on pages x, 2, 12, 93, 96, 97, 99–101, 103, 109, 113, 118).
- [19] An Nguyen, Dominik Krupke, Mary Burbage, Shriya Bhatnagar, Sándor P. Fekete, and Aaron T. Becker. ‘Using a UAV for Destructive Surveys of Mosquito Population’. In: *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*. IEEE, 2018, pp. 7812–7819. doi: [10.1109/ICRA.2018.8463184](https://doi.org/10.1109/ICRA.2018.8463184) (cited on pages x, 118, 123, 126).
- [20] Aaron T. Becker, Mustapha Debboun, Sándor P. Fekete, Dominik Krupke, and An Nguyen. ‘Zapping Zika with a Mosquito-Managing Drone: Computing Optimal Flight Patterns with Minimum Turn Cost (Multimedia Contribution)’. In: *33rd International Symposium on Computational Geometry, SoCG 2017, July 4-7, 2017, Brisbane, Australia*. Ed. by Boris Aronov and Matthew J. Katz. Vol. 77. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 62:1–62:5. doi: [10.4230/LIPIcs.SoCG.2017.62](https://doi.org/10.4230/LIPIcs.SoCG.2017.62) (cited on pages x, 12, 95, 96).
- [21] Sándor P. Fekete, Andreas Haas, Michael Hemmer, Michael Hoffmann, Irina Kostitsyna, Dominik Krupke, Florian Maurer, Joseph S. B. Mitchell, Arne Schmidt, Christiane Schmidt, and Julian Troegel. ‘Computing nonsimple polygons of minimum perimeter’. In: *J. Comput. Geom.* 8.1 (2017), pp. 340–365. doi: [10.20382/jocg.v8i1a13](https://doi.org/10.20382/jocg.v8i1a13) (cited on pages x, 165, 249).

- [22] Arun V. Mahadev, Dominik Krupke, Jan-Marc Reinhardt, Sándor P. Fekete, and Aaron T. Becker. 'Collecting a swarm in a grid environment using shared, global inputs'. In: *IEEE International Conference on Automation Science and Engineering, CASE 2016, Fort Worth, TX, USA, August 21-25, 2016*. IEEE, 2016, pp. 1231–1236. doi: [10.1109/COASE.2016.7743547](https://doi.org/10.1109/COASE.2016.7743547) (cited on pages x, 180, 204, 211, 217).
- [23] Sándor P. Fekete, Andreas Haas, Michael Hemmer, Michael Hoffmann, Irina Kostitsyna, Dominik Krupke, Florian Maurer, Joseph S. B. Mitchell, Arne Schmidt, Christiane Schmidt, and Julian Troegel. 'Computing Nonsimple Polygons of Minimum Perimeter'. In: *Experimental Algorithms - 15th International Symposium, SEA 2016, St. Petersburg, Russia, June 5-8, 2016, Proceedings*. Ed. by Andrew V. Goldberg and Alexander S. Kulikov. Vol. 9685. Lecture Notes in Computer Science. Springer, 2016, pp. 134–149. doi: [10.1007/978-3-319-38851-9_10](https://doi.org/10.1007/978-3-319-38851-9_10) (cited on page x).
- [24] Dominik Krupke, Maximilian Ernestus, Michael Hemmer, and Sándor P. Fekete. 'Distributed cohesive control for robot swarms: Maintaining good connectivity in the presence of exterior forces'. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015, Hamburg, Germany, September 28 - October 2, 2015*. IEEE, 2015, pp. 413–420. doi: [10.1109/IROS.2015.7353406](https://doi.org/10.1109/IROS.2015.7353406) (cited on page x).
- [25] Dominik Krupke, Michael Hemmer, James McLurkin, Yu Zhou, and Sándor P. Fekete. 'A parallel distributed strategy for arraying a scattered robot swarm'. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015, Hamburg, Germany, September 28 - October 2, 2015*. IEEE, 2015, pp. 2795–2802. doi: [10.1109/IROS.2015.7353761](https://doi.org/10.1109/IROS.2015.7353761) (cited on page xi).
- [26] Björn Bankowski, Thimo Clausen, Dirk Ehmen, Maximilian Ernestus, Henning Hasemann, Tobias Jura, Alexander Kröller, Dominik Krupke, and Marco Nikander. 'Panic Room: Experiencing Overload and Having Fun in the Process'. In: *Distributed, Ambient, and Pervasive Interactions - Second International Conference, DAPI 2014, Held as Part of HCI International 2014, Heraklion, Crete, Greece, June 22-27, 2014. Proceedings*. Ed. by Norbert A. Streitz and Panos Markopoulos. Vol. 8530. Lecture Notes in Computer Science. Springer, 2014, pp. 241–252. doi: [10.1007/978-3-319-07788-8_23](https://doi.org/10.1007/978-3-319-07788-8_23) (cited on page xi).
- [27] Dominik Krupke. 'Algorithmic methods for complex dynamic sweeping problems'. Master's thesis. TU Braunschweig, Institute of Operating Systems and Computer Networks, Algorithms Division, 2016 (cited on pages 2, 93, 96, 118, 123, 126, 145).
- [28] Erik D. Demaine, Joseph S. B. Mitchell, and Joseph O'Rourke. *The Open Problems Project*. <http://cs.smith.edu/~orourke/TOPP/>. 2001 (cited on pages 2, 12, 96, 97, 244–246).
- [29] Haje Korth, Michelle F. Thomsen, Karl-Heinz Glassmeier, and W. Scott Phillips. 'Particle tomography of the inner magnetosphere'. In: *Journal of Geophysical Research: Space Physics* 107.A9 (2002), SMP–5 (cited on pages 9, 13, 153, 156).
- [30] Paz Carmi, Matthew J. Katz, Zvi Lotker, and Adi Rosén. 'Connectivity guarantees for wireless networks with directional antennas'. In: *Comput. Geom.* 44.9 (2011), pp. 477–485. doi: [10.1016/j.comgeo.2011.05.003](https://doi.org/10.1016/j.comgeo.2011.05.003) (cited on page 12).
- [31] Rom Aschner and Matthew J. Katz. 'Bounded-Angle Spanning Tree: Modeling Networks with Angular Constraints'. In: *Algorithmica* 77.2 (2017), pp. 349–373. doi: [10.1007/s00453-015-0076-9](https://doi.org/10.1007/s00453-015-0076-9) (cited on page 12).
- [32] Esther M. Arkin, Michael A. Bender, Erik D. Demaine, Sándor P. Fekete, Joseph S. B. Mitchell, and Saurabh Sethia. 'Optimal covering tours with turn costs'. In: *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, January 7-9, 2001, Washington, DC, USA*. Ed. by S. Rao Kosaraju. ACM/SIAM, 2001, pp. 138–147 (cited on pages 12, 97).
- [33] Esther M. Arkin, Michael A. Bender, Erik D. Demaine, Sándor P. Fekete, Joseph S. B. Mitchell, and Saurabh Sethia. 'Optimal Covering Tours with Turn Costs'. In: *SIAM J. Comput.* 35.3 (2005), pp. 531–566. doi: [10.1137/S0097539703434267](https://doi.org/10.1137/S0097539703434267) (cited on pages 12, 97, 102, 103, 109, 113).
- [34] Alok Aggarwal, Don Coppersmith, Sanjeev Khanna, Rajeev Motwani, and Baruch Schieber. 'The Angular-Metric Traveling Salesman Problem'. In: *SIAM J. Comput.* 29.3 (1999), pp. 697–711. doi: [10.1137/S0097539796312721](https://doi.org/10.1137/S0097539796312721) (cited on pages 12, 97, 113).

- [35] Mike Fellows, Panos Giannopoulos, Christian Knauer, Christophe Paul, Frances A. Rosamond, Sue Whitesides, and Nathan Yu. ‘Milling a Graph with Turn Costs: A Parameterized Complexity Perspective’. In: *Graph Theoretic Concepts in Computer Science - 36th International Workshop, WG 2010, Zarós, Crete, Greece, June 28-30, 2010 Revised Papers*. Ed. by Dimitrios M. Thilikos. Vol. 6410. Lecture Notes in Computer Science. 2010, pp. 123–134. doi: [10.1007/978-3-642-16926-7_13](https://doi.org/10.1007/978-3-642-16926-7_13) (cited on page 12).
- [36] Sándor P. Fekete and Gerhard J. Woeginger. ‘Angle-Restricted Tours in the Plane’. In: *Comput. Geom.* 8 (1997), pp. 195–218. doi: [10.1016/S0925-7721\(96\)00012-0](https://doi.org/10.1016/S0925-7721(96)00012-0) (cited on pages 13, 97).
- [37] Richard Loulou. ‘Minimal cut cover of a graph with an application to the testing of electronic boards’. In: *Oper. Res. Lett.* 12.5 (1992), pp. 301–305. doi: [10.1016/0167-6377\(92\)90089-L](https://doi.org/10.1016/0167-6377(92)90089-L) (cited on page 13).
- [38] Rajeev Motwani and Joseph (Seffi) Naor. *On Exact and Approximate Cut Covers of Graphs*. Tech. rep. Stanford, CA, USA: Stanford University, 1994 (cited on pages 13, 15, 34, 41).
- [39] Edna Ayako Hoshino. ‘The minimum cut cover problem’. In: *Electronic Notes in Discrete Mathematics* 37 (2011), pp. 255–260 (cited on page 13).
- [40] Julia Chuzhoy and Sanjeev Khanna. ‘Hardness of cut problems in directed graphs’. In: *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*. Ed. by Jon M. Kleinberg. ACM, 2006, pp. 527–536. doi: [10.1145/1132516.1132593](https://doi.org/10.1145/1132516.1132593) (cited on page 13).
- [41] Ali Allahverdi. ‘The third comprehensive survey on scheduling problems with setup times/costs’. In: *Eur. J. Oper. Res.* 246.2 (2015), pp. 345–378. doi: [10.1016/j.ejor.2015.04.004](https://doi.org/10.1016/j.ejor.2015.04.004) (cited on page 13).
- [42] Ali Allahverdi, Jatinder N.D. Gupta, and Tariq Aldowaisan. ‘A review of scheduling research involving setup considerations’. In: *Omega* 27.2 (1999), pp. 219–239 (cited on page 13).
- [43] Ali Allahverdi, C. T. Ng, T. C. Edwin Cheng, and Mikhail Y. Kovalyov. ‘A survey of scheduling problems with setup times or costs’. In: *Eur. J. Oper. Res.* 187.3 (2008), pp. 985–1032. doi: [10.1016/j.ejor.2006.06.060](https://doi.org/10.1016/j.ejor.2006.06.060) (cited on page 13).
- [44] Yuri N. Sotskov, Alexandre Dolgui, and Frank Werner. ‘Mixed graph coloring for unit-time job-shop scheduling’. In: *International Journal of Mathematical Algorithms* 2.4 (2001), pp. 289–323 (cited on page 13).
- [45] Guoliang Li, Lining Xing, and Yingwu Chen. ‘A hybrid online scheduling mechanism with revision and progressive techniques for autonomous Earth observation satellite’. In: *Acta Astronautica* 140 (2017), pp. 308–321 (cited on pages 13, 81).
- [46] Sean Augenstein, Alejandra Estanislao, Emmanuel Guere, and Sean Blaes. ‘Optimal Scheduling of a Constellation of Earth-Imaging Satellites, for Maximal Data Throughput and Efficient Human Management’. In: *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016*. Ed. by Amanda Jane Coles, Andrew Coles, Stefan Edelkamp, Daniele Magazzeni, and Scott Sanner. AAAI Press, 2016, pp. 345–352 (cited on pages 13, 81).
- [47] Kaoru Watanabe, Masakazu Sengoku, Hiroshi Tamura, and Shoji Shinoda. ‘Cut cover problem in directed graphs’. In: *Proc. of Asia-Pacific Conference on Circuits and Systems. Microelectronics and Integrating Systems (Cat. No.98EX242)*. 1998, pp. 703–706 (cited on page 15).
- [48] Herbert Robbins. ‘A Remark on Stirling’s Formula’. In: *The American Mathematical Monthly* 62.1 (1955), pp. 26–29 (cited on page 16).
- [49] Subhash Khot. ‘Improved Inapproximability Results for MaxClique, Chromatic Number and Approximate Graph Coloring’. In: *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*. IEEE Computer Society, 2001, pp. 600–609. doi: [10.1109/SFCS.2001.959936](https://doi.org/10.1109/SFCS.2001.959936) (cited on page 19).
- [50] László Lovász. ‘Coverings and colorings of hypergraphs’. In: *Southeastern Conf. Combin., Graph Th., Comput. (SEICCGTC)*. 1973, pp. 3–12 (cited on page 23).
- [51] Thomas J. Schaefer. ‘The Complexity of Satisfiability Problems’. In: *Symp. Th. Comp. (STOC)*. 1978, pp. 216–226 (cited on page 23).

- [52] J. A. Hoogeveen. 'Analysis of Christofides' heuristic: Some paths are more difficult than cycles'. In: *Oper. Res. Lett.* 10.5 (1991), pp. 291–295. doi: [10.1016/0167-6377\(91\)90016-I](https://doi.org/10.1016/0167-6377(91)90016-I) (cited on pages 35, 36).
- [53] Rico Zenklusen. 'A 1.5-Approximation for Path TSP'. In: *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*. Ed. by Timothy M. Chan. SIAM, 2019, pp. 1539–1549. doi: [10.1137/1.9781611975482.93](https://doi.org/10.1137/1.9781611975482.93) (cited on pages 35, 36).
- [54] Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. 'A (slightly) improved approximation algorithm for metric TSP'. In: *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*. Ed. by Samir Khuller and Virginia Vassilevska Williams. ACM, 2021, pp. 32–45. doi: [10.1145/3406325.3451009](https://doi.org/10.1145/3406325.3451009) (cited on page 35).
- [55] Jillian Beardwood, John H. Halton, and John Michael Hammersley. 'The shortest path through many points'. In: *Mathematical Proceedings of the Cambridge Philosophical Society*. Vol. 55. 4. Cambridge University Press, 1959, pp. 299–327 (cited on page 35).
- [56] Harold N. Gabow and Herbert H. Westermann. 'Forests, Frames, and Games: Algorithms for Matroid Sums and Applications'. In: *Algorithmica* 7.5&6 (1992), pp. 465–497. doi: [10.1007/BF01758774](https://doi.org/10.1007/BF01758774) (cited on page 36).
- [57] Clair E. Miller, Albert W. Tucker, and Richard A. Zemlin. 'Integer Programming Formulation of Traveling Salesman Problems'. In: *J. ACM* 7.4 (1960), pp. 326–329. doi: [10.1145/321043.321046](https://doi.org/10.1145/321043.321046) (cited on pages 38, 40).
- [58] George B. Dantzig, D. Ray Fulkerson, and Selmer M. Johnson. 'Solution of a Large-Scale Traveling-Salesman Problem'. In: *Oper. Res.* 2.4 (1954), pp. 393–410. doi: [10.1287/opre.2.4.393](https://doi.org/10.1287/opre.2.4.393) (cited on pages 38, 53, 66, 95).
- [59] Daniel Brélaz. 'New methods to color the vertices of a graph'. In: *Comm. ACM* 22.4 (1979), pp. 251–256 (cited on page 41).
- [60] Andrei Novikov. 'PyClustering: Data Mining Library'. In: *J. Open Source Softw.* 4.36 (2019), p. 1230. doi: [10.21105/joss.01230](https://doi.org/10.21105/joss.01230) (cited on page 41).
- [61] M. Gholami, M. Zandieh, and A. Alem-Tabriz. 'Scheduling hybrid flow shop with sequence-dependent setup times and machines with random breakdowns'. In: *Int. J. Adv. Manufact. Tech.* 42.1-2 (2009), pp. 189–201 (cited on page 42).
- [62] Esther M. Arkin, Michael A. Bender, Sándor P. Fekete, Joseph S. B. Mitchell, and Martin Skutella. 'The freeze-tag problem: how to wake up a swarm of robots'. In: *Algorithmica* 46.2 (2006), pp. 193–221 (cited on page 48).
- [63] Esther M. Arkin, Michael A. Bender, and Dongdong Ge. 'Improved approximation algorithms for the freeze-tag problem'. In: *SPAA 2003: Proceedings of the Fifteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, June 7-9, 2003, San Diego, California, USA (part of FCRC 2003)*. Ed. by Arnold L. Rosenberg and Friedhelm Meyer auf der Heide. ACM, 2003, pp. 295–303. doi: [10.1145/777412.777465](https://doi.org/10.1145/777412.777465) (cited on page 48).
- [64] Marcelo O. Sztainberg, Esther M. Arkin, Michael A. Bender, and Joseph S. B. Mitchell. 'Theoretical and experimental analysis of heuristics for the "freeze-tag" robot awakening problem'. In: *IEEE Trans. Robotics* 20.4 (2004), pp. 691–701. doi: [10.1109/TR0.2004.829439](https://doi.org/10.1109/TR0.2004.829439) (cited on page 48).
- [65] Zahra Moezkarimi and Alireza Bagheri. 'A PTAS for geometric 2-FTP'. In: *Inf. Process. Lett.* 114.12 (2014), pp. 670–675. doi: [10.1016/j.ipl.2014.06.017](https://doi.org/10.1016/j.ipl.2014.06.017) (cited on page 48).
- [66] Mikael Hammar, Bengt J. Nilsson, and Mia Persson. 'The Online Freeze-Tag Problem'. In: *LATIN 2006: Theoretical Informatics, 7th Latin American Symposium, Valdivia, Chile, March 20-24, 2006, Proceedings*. Ed. by José R. Correa, Alejandro Hevia, and Marcos A. Kiwi. Vol. 3887. Lecture Notes in Computer Science. Springer, 2006, pp. 569–579. doi: [10.1007/11682462_53](https://doi.org/10.1007/11682462_53) (cited on page 48).
- [67] Anatole Beck and D. J. Newman. 'Yet more on the linear search problem'. In: *Israel Journal of Mathematics* 8.4 (Dec. 1970), pp. 419–429. doi: [10.1007/BF02798690](https://doi.org/10.1007/BF02798690) (cited on page 52).

- [68] Prosenjit Bose, Jean-Lou De Carufel, and Stephane Durocher. 'Searching on a line: A complete characterization of the optimal solution'. In: *Theor. Comput. Sci.* 569 (2015), pp. 24–42. doi: [10.1016/j.tcs.2014.12.007](https://doi.org/10.1016/j.tcs.2014.12.007) (cited on page 52).
- [69] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2021. URL: <https://www.gurobi.com> (cited on page 52).
- [70] Laurent Perron and Vincent Furnon. *OR-Tools*. Version 9.0. Google (cited on pages 52, 198).
- [71] Emilie Danna, Edward Rothberg, and Claude Le Pape. 'Exploring relaxation induced neighborhoods to improve MIP solutions'. In: *Math. Program.* 102.1 (2005), pp. 71–90. doi: [10.1007/s10107-004-0518-7](https://doi.org/10.1007/s10107-004-0518-7) (cited on page 64).
- [72] Laurent Perron and Frédéric Didier. 'Master Class Session on CP-SAT'. In: *17th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2020)*. Video available at <https://www.youtube.com/watch?v=lmy1ddn4cyw>. 2020 (cited on page 65).
- [73] Stephen A. Cook. 'The Complexity of Theorem-Proving Procedures'. In: *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*. Ed. by Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman. ACM, 1971, pp. 151–158. doi: [10.1145/800157.805047](https://doi.org/10.1145/800157.805047) (cited on page 65).
- [74] Olga Ohrimenko, Peter J. Stuckey, and Michael Codish. 'Propagation via lazy clause generation'. In: *Constraints An Int. J.* 14.3 (2009), pp. 357–391. doi: [10.1007/s10601-008-9064-x](https://doi.org/10.1007/s10601-008-9064-x) (cited on page 66).
- [75] Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability*. Addison-Wesley Professional, 2015 (cited on page 66).
- [76] Armin Biere, Marijn Heule, and Hans van Maaren. *Handbook of satisfiability*. Vol. 185. IOS press, 2009 (cited on page 66).
- [77] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. 'Chaff: Engineering an Efficient SAT Solver'. In: *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*. ACM, 2001, pp. 530–535. doi: [10.1145/378239.379017](https://doi.org/10.1145/378239.379017) (cited on page 67).
- [78] William J. Cook, Collette R. Coullard, and György Turán. 'On the complexity of cutting-plane proofs'. In: *Discret. Appl. Math.* 18.1 (1987), pp. 25–38. doi: [10.1016/0166-218X\(87\)90039-4](https://doi.org/10.1016/0166-218X(87)90039-4) (cited on page 67).
- [79] Duana L. Bindschadler, Carole A. Boyles, Carlos Carrion, and Chris L. Delp. 'MOS 2.0: The next generation in mission operations systems'. In: *SpaceOps Conference*. NASA, 2010 (cited on page 79).
- [80] Michael L. Pinedo. *Scheduling: theory, algorithms, and systems*. Springer, 2016 (cited on page 81).
- [81] Sara C. Spangelo. 'Modeling and Optimizing Space Networks for Improved Communication Capacity.' PhD thesis. 2013 (cited on page 81).
- [82] Dick Stottler. 'Satellite communication scheduling, optimization, and deconfliction using artificial intelligence techniques'. In: *AIAA Infotech@ Aerospace 2010*. 2010, p. 3424 (cited on page 81).
- [83] Fabrizio Marinelli, Salvatore Nocella, Fabrizio Rossi, and Stefano Smriglio. 'A Lagrangian heuristic for satellite range scheduling with resource constraints'. In: *Comput. Oper. Res.* 38.11 (2011), pp. 1572–1583. doi: [10.1016/j.cor.2011.01.016](https://doi.org/10.1016/j.cor.2011.01.016) (cited on page 81).
- [84] Junghyun Lee, Haedong Kim, Hyun Chung, and Kwanghee Ko. 'Genetic algorithm-based scheduling for ground support of multiple satellites and antennae considering operation modes'. In: *International Journal of Aeronautical and Space Sciences* 17.1 (2016), pp. 89–100 (cited on pages 81, 84).
- [85] Michael P. Wellman, William E. Walsh, Peter R. Wurman, and Jeffrey K. MacKie-Mason. 'Auction Protocols for Decentralized Scheduling'. In: *Games Econ. Behav.* 35.1-2 (2001), pp. 271–303. doi: [10.1006/game.2000.0822](https://doi.org/10.1006/game.2000.0822) (cited on page 82).
- [86] Stephen J. Rassenti, Vernon L. Smith, and Robert L. Bulfin. 'A combinatorial auction mechanism for airport time slot allocation'. In: *The Bell Journal of Economics* (1982), pp. 402–417 (cited on page 82).
- [87] Michael H. Rothkopf, Aleksandar Pekeč, and Ronald M. Harstad. 'Computationally Manageable Combinational Auctions'. In: *Management Science* 44.8 (1998), pp. 1131–1147. doi: [10.1287/mnsc.44.8.1131](https://doi.org/10.1287/mnsc.44.8.1131) (cited on page 82).

- [88] Yossi Sheffi. 'Combinatorial Auctions in the Procurement of Transportation Services'. In: *Interfaces* 34.4 (2004), pp. 245–252. doi: [10.1287/inte.1040.0075](https://doi.org/10.1287/inte.1040.0075) (cited on page 82).
- [89] Sourav Pal, Sumantra R. Kundu, Mainak Chatterjee, and Sajal K. Das. 'Combinatorial reverse auction based scheduling in multirate wireless systems'. In: *IEEE Transactions on Computers* 56.10 (2007), pp. 1329–1341. doi: [10.1109/TC.2007.1082](https://doi.org/10.1109/TC.2007.1082) (cited on page 82).
- [90] Hassan Harb, Jan Niklas Paprott, Peter Matthes, Thomas Schütz, Rita Streblov, and Dirk Müller. 'Decentralized scheduling strategy of heating systems for balancing the residual load'. In: *Building and Environment* 86 (2015), pp. 132–140. doi: [10.1016/j.buildenv.2014.12.015](https://doi.org/10.1016/j.buildenv.2014.12.015) (cited on page 82).
- [91] Michael P. Wellman. 'A Market-Oriented Programming Environment and its Application to Distributed Multicommodity Flow Problems'. In: *J. Artif. Intell. Res.* 1 (1993), pp. 1–23. doi: [10.1613/jair.2](https://doi.org/10.1613/jair.2) (cited on page 82).
- [92] Giuseppe Confessore, Stefano Giordani, and Silvia Rismondo. 'A market-based multi-agent system model for decentralized multi-project scheduling'. In: *Ann. Oper. Res.* 150.1 (2007), pp. 115–135. doi: [10.1007/s10479-006-0158-9](https://doi.org/10.1007/s10479-006-0158-9) (cited on page 82).
- [93] William E. Walsh, Michael P. Wellman, Peter R. Wurman, and Jeffrey K. MacKie-Mason. 'Some Economics of Market-Based Distributed Scheduling'. In: *Proceedings of the 18th International Conference on Distributed Computing Systems, Amsterdam, The Netherlands, May 26-29, 1998*. IEEE Computer Society, 1998, pp. 612–621. doi: [10.1109/ICDCS.1998.679848](https://doi.org/10.1109/ICDCS.1998.679848) (cited on page 82).
- [94] Vikas Kumar, Shashi Kumar, M. K. Tiwari, and F. T.S. Chan. 'Auction-based approach to resolve the scheduling problem in the steel making process'. In: *International Journal of Production Research* 44.8 (2006), pp. 1503–1522. doi: [10.1080/00207540500434713](https://doi.org/10.1080/00207540500434713) (cited on page 82).
- [95] Stefano Giordani, Marin Lujak, and Francesco Martinelli. 'A distributed multi-agent production planning and scheduling framework for mobile robots'. In: *Comput. Ind. Eng.* 64.1 (2013), pp. 19–30. doi: [10.1016/j.cie.2012.09.004](https://doi.org/10.1016/j.cie.2012.09.004) (cited on page 82).
- [96] Qing lin Guo and Ming Zhang. 'An agent-oriented approach to resolve scheduling optimization in intelligent manufacturing'. In: *Robotics and Computer-Integrated Manufacturing* 26.1 (2010), pp. 39–45. doi: [10.1016/j.rcim.2009.02.003](https://doi.org/10.1016/j.rcim.2009.02.003) (cited on page 82).
- [97] David C. Parkes and Lyle H. Ungar. 'An auction-based method for decentralized train scheduling'. In: *Proceedings of the Fifth International Conference on Autonomous Agents, AGENTS 2001, Montreal, Canada, May 28 - June 1, 2001*. Ed. by Elisabeth André, Sandip Sen, Claude Frasson, and Jörg P. Müller. ACM, 2001, pp. 43–50. doi: [10.1145/375735.375866](https://doi.org/10.1145/375735.375866) (cited on page 82).
- [98] Ralf Borndörfer, Martin Grötschel, Sascha Lukac, Kay Mitusch, Thomas Schlechte, Sören Schultz, and Andreas Tanner. 'An auctioning approach to railway slot allocation'. In: *Competition and Regulation in Network Industries* 1.2 (2006), pp. 163–197 (cited on page 82).
- [99] Stephen J. Rassenti, Vernon L. Smith, and Robert L. Bulfin. 'A combinatorial auction mechanism for airport time slot allocation'. In: *Handbook of Spectrum Auction Design*. 2017, pp. 373–390. doi: [10.1017/9781316471609.019](https://doi.org/10.1017/9781316471609.019) (cited on page 82).
- [100] Paul Davidsson, Lawrence Henesey, Linda Ramstedt, Johanna Törnquist, and Fredrik Wernstedt. *An analysis of agent-based approaches to transport logistics*. 2005. doi: [10.1016/j.trc.2005.07.002](https://doi.org/10.1016/j.trc.2005.07.002) (cited on page 82).
- [101] Daniel M. Reeves, Michael P. Wellman, Jeffrey K. MacKie-Mason, and Anna Osepayshvili. 'Exploring bidding strategies for market-based scheduling'. In: *Decis. Support Syst.* 39.1 (2005), pp. 67–85. doi: [10.1016/j.dss.2004.08.014](https://doi.org/10.1016/j.dss.2004.08.014) (cited on page 82).
- [102] David C. Parkes. 'iBundle: an efficient ascending price bundle auction'. In: *Proceedings of the First ACM Conference on Electronic Commerce (EC-99), Denver, CO, USA, November 3-5, 1999*. Ed. by Stuart I. Feldman and Michael P. Wellman. ACM, 1999, pp. 148–157. doi: [10.1145/336992.337032](https://doi.org/10.1145/336992.337032) (cited on page 82).
- [103] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005 (cited on page 85).

- [104] David L. Applegate, Robert E. Bixby, Vašek Chvátal, and William J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2007 (cited on page 95).
- [105] Esther M. Arkin, Sándor P. Fekete, and Joseph S. B. Mitchell. ‘Approximation algorithms for lawn mowing and milling’. In: *Comput. Geom.* 17.1-2 (2000), pp. 25–50. doi: [10.1016/S0925-7721\(00\)00015-8](https://doi.org/10.1016/S0925-7721(00)00015-8) (cited on pages 97, 98).
- [106] Simeon C. Ntafos. ‘Watchman Routes Under Limited Visibility’. In: *Comput. Geom.* 1 (1991), pp. 149–170. doi: [10.1016/0925-7721\(92\)90014-J](https://doi.org/10.1016/0925-7721(92)90014-J) (cited on page 97).
- [107] Sylvain Lazard, John Reif, and Hongyan Wang. ‘The complexity of the two dimensional curvature-constrained shortest-path problem’. In: *Proceedings of the 3rd International Workshop on the Algorithmic Foundations of Robotics (WAFR)*. 1998, pp. 49–57 (cited on page 97).
- [108] Lester E. Dubins. ‘On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents’. In: *American Journal of Mathematics* 79.3 (1957), pp. 497–516 (cited on page 97).
- [109] Jean-Daniel Boissonnat and Sylvain Lazard. ‘A Polynomial-Time Algorithm for Computing a Shortest Path of Bounded Curvature Amidst Moderate Obstacles (Extended Abstract)’. In: *Proceedings of the Twelfth Annual Symposium on Computational Geometry, Philadelphia, PA, USA, May 24-26, 1996*. Ed. by Sue Whitesides. ACM, 1996, pp. 242–251. doi: [10.1145/237218.237393](https://doi.org/10.1145/237218.237393) (cited on page 97).
- [110] Pankaj K. Agarwal, Therese C. Biedl, Sylvain Lazard, Steve Robbins, Subhash Suri, and Sue Whitesides. ‘Curvature-Constrained Shortest Paths in a Convex Polygon’. In: *SIAM J. Comput.* 31.6 (2002), pp. 1814–1851. doi: [10.1137/S0097539700374550](https://doi.org/10.1137/S0097539700374550) (cited on page 97).
- [111] Pankaj K. Agarwal and Hongyan Wang. ‘Approximation Algorithms for Curvature-Constrained Shortest Paths’. In: *SIAM J. Comput.* 30.6 (2000), pp. 1739–1772. doi: [10.1137/S0097539796307790](https://doi.org/10.1137/S0097539796307790) (cited on page 97).
- [112] Ryo Takei, Richard Tsai, Haochong Shen, and Yanina Landa. ‘A practical path-planning algorithm for a simple car: A Hamilton-Jacobi approach’. In: *Proceedings of the 29th American Control Conference (ACC)*. 2010, pp. 6175–6180 (cited on page 97).
- [113] Jerome Le Ny, Eric Feron, and Emilio Frazzoli. ‘On the Dubins Traveling Salesman Problem’. In: *IEEE Trans. Autom. Control.* 57.1 (2012), pp. 265–270. doi: [10.1109/TAC.2011.2166311](https://doi.org/10.1109/TAC.2011.2166311) (cited on page 97).
- [114] Olaf Maurer. ‘Winkelminimierung bei Überdeckungsproblemen in Graphen’. Diplomarbeit. Technische Universität Berlin, 2009 (cited on page 97).
- [115] Igor R. de Assis and Cid C. de Souza. ‘Experimental Evaluation of Algorithms for the Orthogonal Milling Problem with Turn Costs’. In: *Experimental Algorithms - 10th International Symposium, SEA 2011, Kolimpari, Chania, Crete, Greece, May 5-7, 2011. Proceedings*. Ed. by Panos M. Pardalos and Steffen Rebennack. Vol. 6630. Lecture Notes in Computer Science. Springer, 2011, pp. 304–314. doi: [10.1007/978-3-642-20662-7_26](https://doi.org/10.1007/978-3-642-20662-7_26) (cited on pages 97, 109, 113).
- [116] Gerold Jäger and Paul Molitor. ‘Algorithms and Experimental Study for the Traveling Salesman Problem of Second Order’. In: *Combinatorial Optimization and Applications, Second International Conference, COCOA 2008, St. John’s, NL, Canada, August 21-24, 2008. Proceedings*. Ed. by Boting Yang, Ding-Zhu Du, and Cao An Wang. Vol. 5165. Lecture Notes in Computer Science. Springer, 2008, pp. 211–224. doi: [10.1007/978-3-540-85097-7_20](https://doi.org/10.1007/978-3-540-85097-7_20) (cited on page 98).
- [117] Borzou Rostami, Federico Malucelli, Pietro Belotti, and Stefano Gualandi. ‘Quadratic TSP: A lower bounding procedure and a column generation approach’. In: *Proceedings of the 2013 Federated Conference on Computer Science and Information Systems, Kraków, Poland, September 8-11, 2013*. Ed. by Maria Ganzha, Leszek A. Maciaszek, and Marcin Paprzycki. 2013, pp. 377–384 (cited on page 98).
- [118] Oswin Aichholzer, Anja Fischer, Frank Fischer, J. Fabian Meier, Ulrich Pferschy, Alexander Pilz, and Rostislav Staněk. ‘Minimization and maximization versions of the quadratic travelling salesman problem’. In: *Optimization* 66.4 (2017), pp. 521–546 (cited on page 98).
- [119] Alexander Zelinsky, Ray A. Jarvis, J. C. Byrne, and Shin’ichi Yuta. ‘Planning paths of complete coverage of an unstructured environment by a mobile robot’. In: *Proceedings of the 6th International Conference on Advanced Robotics (ICAR)*. 1993, pp. 533–538 (cited on page 98).

- [120] Yoav Gabriely and Elon Rimon. 'Spiral-STC: An On-Line Coverage Algorithm of Grid Environments by a Mobile Robot'. In: *Proceedings of the 2002 IEEE International Conference on Robotics and Automation, ICRA 2002, May 11-15, 2002, Washington, DC, USA*. IEEE, 2002, pp. 954–960. doi: [10.1109/ROBOT.2002.1013479](https://doi.org/10.1109/ROBOT.2002.1013479) (cited on page 98).
- [121] Wesley H. Huang. 'Optimal Line-sweep-based Decompositions for Coverage Algorithms'. In: *Proceedings of the 2001 IEEE International Conference on Robotics and Automation, ICRA 2001, May 21-26, 2001, Seoul, Korea*. IEEE, 2001, pp. 27–32. doi: [10.1109/ROBOT.2001.932525](https://doi.org/10.1109/ROBOT.2001.932525) (cited on page 98).
- [122] Zhiyang Yao, Satyandra K. Gupta, and Dana S. Nau. 'Hybrid cutter path generation for 2.5D milling operation'. In: *Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIE)*. 2002, pp. 703–714 (cited on page 98).
- [123] Osman Ali, Bart Verlinden, and Dirk Van Oudheusden. 'Infield logistics planning for crop-harvesting operations'. In: *Engineering Optimization* 41.2 (2009), pp. 183–197 (cited on page 98).
- [124] Ali Ahmadzadeh, James F. Keller, George J. Pappas, Ali Jadbabaie, and Vijay Kumar. 'An Optimization-Based Approach to Time-Critical Cooperative Surveillance and Coverage with UAVs'. In: *Experimental Robotics, The 10th International Symposium on Experimental Robotics [ISER '06, July 6-10, 2006, Rio de Janeiro, Brazil]*. Ed. by Oussama Khatib, Vijay Kumar, and Daniela Rus. Vol. 39. Springer Tracts in Advanced Robotics. Springer, 2006, pp. 491–500. doi: [10.1007/978-3-540-77457-0_46](https://doi.org/10.1007/978-3-540-77457-0_46) (cited on page 98).
- [125] Amit Agarwal, Meng-Hiot Lim, Meng Joo Er, and Chan Yee Chew. 'ACO for a new TSP in region coverage'. In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, Edmonton, Alberta, Canada, August 2-6, 2005*. IEEE, 2005, pp. 1717–1722. doi: [10.1109/IR05.2005.1545460](https://doi.org/10.1109/IR05.2005.1545460) (cited on page 98).
- [126] Giorgio Ausiello, Vincenzo Bonifaci, Stefano Leonardi, and Alberto Marchetti-Spaccamela. 'Prize Collecting Traveling Salesman and Related Problems'. In: *Handbook of Approximation Algorithms and Metaheuristics, Second Edition, Volume 1: Methodologies and Traditional Applications*. Ed. by Teofilo F. Gonzalez. Chapman and Hall/CRC, 2018, pp. 611–628. doi: [10.1201/9781351236423-34](https://doi.org/10.1201/9781351236423-34) (cited on page 98).
- [127] Keld Helsgaun. 'Solving the equality generalized traveling salesman problem using the Lin-Kernighan-Helsgaun Algorithm'. In: *Math. Program. Comput.* 7.3 (2015), pp. 269–287. doi: [10.1007/s12532-015-0080-8](https://doi.org/10.1007/s12532-015-0080-8) (cited on page 98).
- [128] Vladimir Kolmogorov. 'Blossom V: a new implementation of a minimum cost perfect matching algorithm'. In: *Math. Program. Comput.* 1.1 (2009), pp. 43–67. doi: [10.1007/s12532-009-0002-8](https://doi.org/10.1007/s12532-009-0002-8) (cited on pages 104, 121).
- [129] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. 'A Nearly-Linear Time Framework for Graph-Structured Sparsity'. In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*. Ed. by Subbarao Kambhampati. IJCAI/AAAI Press, 2016, pp. 4165–4169 (cited on page 104).
- [130] Tauã M. Cabreira, Lisane B. Brisolara, and Paulo R. Ferreira Jr. 'Survey on Coverage Path Planning with Unmanned Aerial Vehicles'. In: *Drones* 3.1 (2019). doi: [10.3390/drones3010004](https://doi.org/10.3390/drones3010004) (cited on pages 109, 113).
- [131] Richard Bormann, Florian Jordan, Joshua Hampp, and Martin Hägele. 'Indoor Coverage Path Planning: Survey, Implementation, Analysis'. In: *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*. IEEE, 2018, pp. 1718–1725. doi: [10.1109/ICRA.2018.8460566](https://doi.org/10.1109/ICRA.2018.8460566) (cited on pages 109, 113, 128).
- [132] S. Marshall and J. G. Griffiths. 'A survey of cutter path construction techniques for milling machines'. In: *The International Journal of Production Research* 32.12 (1994), pp. 2861–2877 (cited on page 109).
- [133] Ibrahim A. Hameed. 'Intelligent Coverage Path Planning for Agricultural Robots and Autonomous Machines on Three-Dimensional Terrain'. In: *J. Intell. Robot. Syst.* 74.3-4 (2014), pp. 965–983. doi: [10.1007/s10846-013-9834-6](https://doi.org/10.1007/s10846-013-9834-6) (cited on pages 109, 114).

- [134] David Applegate, Robert Bixby, Vasek Chvatal, and William Cook. *Concorde TSP solver*. URL: <http://www.math.uwaterloo.ca/tsp/concorde.html> (cited on pages 109, 234).
- [135] David L. Applegate, Robert E. Bixby, Vasek Chvátal, William J. Cook, Daniel G. Espinoza, Marcos Goycoolea, and Keld Helsgaun. 'Certification of an optimal TSP tour through 85,900 cities'. In: *Oper. Res. Lett.* 37.1 (2009), pp. 11–15. doi: [10.1016/j.orl.2008.09.006](https://doi.org/10.1016/j.orl.2008.09.006) (cited on pages 109, 113).
- [136] Howie Choset. 'Coverage for robotics - A survey of recent results'. In: *Ann. Math. Artif. Intell.* 31.1-4 (2001), pp. 113–126. doi: [10.1023/A:1016639210559](https://doi.org/10.1023/A:1016639210559) (cited on page 113).
- [137] Enric Galceran and Marc Carreras. 'A survey on coverage path planning for robotics'. In: *Robotics Auton. Syst.* 61.12 (2013), pp. 1258–1276. doi: [10.1016/j.robot.2013.09.004](https://doi.org/10.1016/j.robot.2013.09.004) (cited on page 113).
- [138] Timo Oksanen and Arto Visala. 'Coverage path planning algorithms for agricultural field machines'. In: *J. Field Robotics* 26.8 (2009), pp. 651–668. doi: [10.1002/rob.20300](https://doi.org/10.1002/rob.20300) (cited on pages 113, 114).
- [139] Howie Choset and Philippe Pignon. 'Coverage path planning: The boustrophedon cellular decomposition'. In: *Field and service robotics*. Springer, 1998, pp. 203–209 (cited on page 113).
- [140] Howie Choset, Ercan U. Acar, Alfred A. Rizzi, and Jonathan E. Luntz. 'Exact Cellular Decompositions in Terms of Critical Points of Morse Functions'. In: *Proceedings of the 2000 IEEE International Conference on Robotics and Automation, ICRA 2000, April 24-28, 2000, San Francisco, CA, USA*. IEEE, 2000, pp. 2270–2277. doi: [10.1109/ROBOT.2000.846365](https://doi.org/10.1109/ROBOT.2000.846365) (cited on page 113).
- [141] Richard Bormann, Joshua Hampp, and Martin Hägele. 'New brooms sweep clean - an autonomous robotic cleaning assistant for professional office cleaning'. In: *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*. IEEE, 2015, pp. 4470–4477. doi: [10.1109/ICRA.2015.7139818](https://doi.org/10.1109/ICRA.2015.7139818) (cited on pages 113, 114).
- [142] Ghulam Murtaza, Salil S. Kanhere, and Sanjay K. Jha. 'Priority-based coverage path planning for Aerial Wireless Sensor Networks'. In: *2013 IEEE Eighth International Conference on Intelligent Sensors, Sensor Networks and Information Processing, Melbourne, Australia, April 2-5, 2013*. IEEE, 2013, pp. 219–224. doi: [10.1109/ISSNIP.2013.6529792](https://doi.org/10.1109/ISSNIP.2013.6529792) (cited on page 113).
- [143] Xiaoming Zheng, Sven Koenig, David Kempe, and Sonal Jain. 'Multirobot Forest Coverage for Weighted and Unweighted Terrain'. In: *IEEE Trans. Robotics* 26.6 (2010), pp. 1018–1031. doi: [10.1109/TR0.2010.2072271](https://doi.org/10.1109/TR0.2010.2072271) (cited on pages 113, 114).
- [144] Gokarna Sharma, Ayan Dutta, and Jong-Hoon Kim. 'Optimal Online Coverage Path Planning with Energy Constraints'. In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019*. Ed. by Edith Elkind, Manuela Veloso, Noa Agmon, and Matthew E. Taylor. International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 1189–1197 (cited on pages 113, 114).
- [145] Jalil Modares, Farshad Ghanei, Nicholas Mastrorarde, and Karthik Dantu. 'UB-ANC planner: Energy efficient coverage path planning with multiple drones'. In: *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*. IEEE, 2017, pp. 6182–6189. doi: [10.1109/ICRA.2017.7989732](https://doi.org/10.1109/ICRA.2017.7989732) (cited on pages 113, 114).
- [146] Alon Itai, Christos H. Papadimitriou, and Jayme Luiz Szwarcfiter. 'Hamilton Paths in Grid Graphs'. In: *SIAM J. Comput.* 11.4 (1982), pp. 676–686. doi: [10.1137/0211056](https://doi.org/10.1137/0211056) (cited on page 113).
- [147] David L. Applegate, Robert E. Bixby, Vašek Chvátal, and William J. Cook. *The traveling salesman problem*. Princeton university press, 2011 (cited on page 113).
- [148] Anja Fischer, Frank Fischer, Gerold Jäger, Jens Keilwagen, Paul Molitor, and Ivo Grosse. 'Exact algorithms and heuristics for the Quadratic Traveling Salesman Problem with an application in bioinformatics'. In: *Discret. Appl. Math.* 166 (2014), pp. 97–114. doi: [10.1016/j.dam.2013.09.011](https://doi.org/10.1016/j.dam.2013.09.011) (cited on page 113).
- [149] Katharin R. Jensen-Nau, Tucker Hermans, and Kam K. Leang. 'Near-Optimal Area-Coverage Path Planning of Energy-Constrained Aerial Robots With Application in Autonomous Environmental Monitoring'. In: *IEEE Trans Autom. Sci. Eng.* 18.3 (2021), pp. 1453–1468. doi: [10.1109/TASE.2020.3016276](https://doi.org/10.1109/TASE.2020.3016276) (cited on page 114).

- [150] Daniel E. Soltero, Mac Schwager, and Daniela Rus. ‘Decentralized path planning for coverage tasks using gradient descent adaptive control’. In: *Int. J. Robotics Res.* 33.3 (2014), pp. 401–425. doi: [10.1177/0278364913497241](https://doi.org/10.1177/0278364913497241) (cited on page 114).
- [151] Christos Papachristos, Kostas Alexis, Luis Rodolfo Garcia Carrillo, and Anthony Tzes. ‘Distributed infrastructure inspection path planning for aerial robotics subject to time constraints’. In: *2016 international conference on unmanned aircraft systems (ICUAS)*. IEEE, 2016, pp. 406–412 (cited on page 114).
- [152] Kai Olav Ellefsen, Herman Augusto Lepikson, and Jan C. Albiez. ‘Multiobjective coverage path planning: Enabling automated inspection of complex, real-world structures’. In: *Appl. Soft Comput.* 61 (2017), pp. 264–282. doi: [10.1016/j.asoc.2017.07.051](https://doi.org/10.1016/j.asoc.2017.07.051) (cited on page 114).
- [153] Taua M. Cabreira, Carmelo Di Franco, Paulo Roberto Ferreira Jr., and Giorgio C. Buttazzo. ‘Energy-Aware Spiral Coverage Path Planning for UAV Photogrammetric Applications’. In: *IEEE Robotics Autom. Lett.* 3.4 (2018), pp. 3662–3668. doi: [10.1109/LRA.2018.2854967](https://doi.org/10.1109/LRA.2018.2854967) (cited on page 114).
- [154] Carmelo Di Franco and Giorgio C. Buttazzo. ‘Coverage Path Planning for UAVs Photogrammetry with Energy and Resolution Constraints’. In: *J. Intell. Robotic Syst.* 83.3-4 (2016), pp. 445–462. doi: [10.1007/s10846-016-0348-x](https://doi.org/10.1007/s10846-016-0348-x) (cited on page 114).
- [155] John Ware and Nicholas Roy. ‘An analysis of wind field estimation and exploitation for quadrotor flight in the urban canopy layer’. In: *2016 IEEE International Conference on Robotics and Automation, ICRA 2016, Stockholm, Sweden, May 16-21, 2016*. Ed. by Danica Kragic, Antonio Bicchi, and Alessandro De Luca. IEEE, 2016, pp. 1507–1514. doi: [10.1109/ICRA.2016.7487287](https://doi.org/10.1109/ICRA.2016.7487287) (cited on page 114).
- [156] Joseph S. B. Mitchell and Christos H. Papadimitriou. ‘The Weighted Region Problem: Finding Shortest Paths Through a Weighted Planar Subdivision’. In: *J. ACM* 38.1 (1991), pp. 18–73. doi: [10.1145/102782.102784](https://doi.org/10.1145/102782.102784) (cited on page 114).
- [157] Jozef Vörös. ‘Mobile Robot Path Planning Among Weighted Regions Using Quadtree Representations’. In: *Computer Aided Systems Theory - EUROCAST’99, Vienna, Austria, September 29 - October 2, 1999, Proceedings*. Ed. by Franz Pichler, Roberto Moreno-Díaz, and Peter Kopacek. Vol. 1798. Lecture Notes in Computer Science. Springer, 1999, pp. 239–249. doi: [10.1007/10720123_20](https://doi.org/10.1007/10720123_20) (cited on page 114).
- [158] Neil C. Rowe and Robert S. Alexander. ‘Finding Optimal-Path Maps for Path Planning across Weighted Regions’. In: *Int. J. Robotics Res.* 19.2 (2000), pp. 83–95. doi: [10.1177/02783640022066761](https://doi.org/10.1177/02783640022066761) (cited on page 114).
- [159] L. H. Nam, Loulin Huang, Xue Jun Li, and Jianfeng Xu. ‘An approach for coverage path planning for UAVs’. In: *IEEE 14th International Workshop on Advanced Motion Control, AMC 2016, Auckland, New Zealand, April 22-24, 2016*. IEEE, 2016, pp. 411–416. doi: [10.1109/AMC.2016.7496385](https://doi.org/10.1109/AMC.2016.7496385) (cited on page 115).
- [160] Oleksandr Artemenko, Omachonu Joshua Dominic, Oleksandr Andreyev, and Andreas Mitschele-Thiel. ‘Energy-Aware Trajectory Planning for the Localization of Mobile Devices Using an Unmanned Aerial Vehicle’. In: *25th International Conference on Computer Communication and Networks, ICCCN 2016, Waikoloa, HI, USA, August 1-4, 2016*. IEEE, 2016, pp. 1–9. doi: [10.1109/ICCCN.2016.7568517](https://doi.org/10.1109/ICCCN.2016.7568517) (cited on page 115).
- [161] Erik J. Forsmo, Esten I. Grøtli, Thor I. Fossen, and Tor A. Johansen. ‘Optimal search mission with unmanned aerial vehicles using mixed integer linear programming’. In: *2013 International conference on unmanned aircraft systems (ICUAS)*. IEEE, 2013, pp. 253–259 (cited on page 115).
- [162] Ralph E. Gomory. ‘Outline of an Algorithm for Integer Solutions to Linear Programs and An Algorithm for the Mixed Integer Problem’. In: *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*. Ed. by Michael Jünger, Thomas M. Lieblich, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey. Springer, 2010, pp. 77–103. doi: [10.1007/978-3-540-68279-0_4](https://doi.org/10.1007/978-3-540-68279-0_4) (cited on page 118).
- [163] Michel X. Goemans and David P. Williamson. ‘A General Approximation Technique for Constrained Forest Problems’. In: *SIAM J. Comput.* 24.2 (1995), pp. 296–317. doi: [10.1137/S0097539793242618](https://doi.org/10.1137/S0097539793242618) (cited on page 125).

- [164] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. ‘A Nearly-Linear Time Framework for Graph-Structured Sparsity’. In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. Ed. by Francis R. Bach and David M. Blei. Vol. 37. JMLR Workshop and Conference Proceedings. JMLR.org, 2015, pp. 928–937 (cited on page 125).
- [165] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. ‘A fast, adaptive variant of the Goemans-Williamson scheme for the prize-collecting Steiner tree problem’. In: *Work. 11th DIMACS Implement. Chall* (2014) (cited on page 125).
- [166] Michael Perk. ‘Theoretical and practical approaches for optimizing lawn mowing and milling’. MA thesis. Institute of Operating Systems and Computer Networks, 2021 (cited on page 129).
- [167] Marshall W. Bern and Paul E. Plassmann. ‘Mesh Generation’. In: *Handbook of Computational Geometry*. Ed. by Jörg-Rüdiger Sack and Jorge Urrutia. North Holland / Elsevier, 2000, pp. 291–332. doi: [10.1016/b978-044482537-7/50007-3](https://doi.org/10.1016/b978-044482537-7/50007-3) (cited on page 134).
- [168] Marshall W. Bern. ‘Triangulations and Mesh Generation’. In: *Handbook of Discrete and Computational Geometry, Second Edition*. Ed. by Jacob E. Goodman and Joseph O’Rourke. Chapman and Hall/CRC, 2004, pp. 563–582. doi: [10.1201/9781420035315.ch25](https://doi.org/10.1201/9781420035315.ch25) (cited on page 134).
- [169] Christophe Geuzaine and Jean-François Remacle. ‘Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities’. In: *International journal for numerical methods in engineering* 79.11 (2009), pp. 1309–1331 (cited on pages 134–136).
- [170] David H. Douglas and Thomas K. Peucker. ‘Algorithms for the reduction of the number of points required to represent a digitized line or its caricature’. In: *Cartographica: the international journal for geographic information and geovisualization* 10.2 (1973), pp. 112–122 (cited on pages 134, 135).
- [171] Qiang Du, Vance Faber, and Max D. Gunzburger. ‘Centroidal Voronoi Tessellations: Applications and Algorithms’. In: *SIAM Rev.* 41.4 (1999), pp. 637–676. doi: [10.1137/S0036144599352836](https://doi.org/10.1137/S0036144599352836) (cited on pages 135, 138).
- [172] Stuart P. Lloyd. ‘Least squares quantization in PCM’. In: *IEEE Trans. Inf. Theory* 28.2 (1982), pp. 129–136. doi: [10.1109/TIT.1982.1056489](https://doi.org/10.1109/TIT.1982.1056489) (cited on page 135).
- [173] Long Chen and Michael Holst. ‘Efficient mesh optimization schemes based on optimal Delaunay triangulations’. In: *Computer Methods in Applied Mechanics and Engineering* 200.9-12 (2011), pp. 967–984 (cited on page 135).
- [174] Nico Schlömer and Adam Dobrawa. *nschloe/optimesh: None*. Version v0.8.2. 2021. doi: [10.5281/zenodo.4728056](https://doi.org/10.5281/zenodo.4728056) (cited on page 135).
- [175] Nico Schlömer and J. Hariharan. *nschloe/dmsh: None*. Version 0.2.17. 2021. doi: [10.5281/zenodo.5019221](https://doi.org/10.5281/zenodo.5019221) (cited on page 136).
- [176] Per-Olof Persson and Gilbert Strang. ‘A Simple Mesh Generator in MATLAB’. In: *SIAM Rev.* 46.2 (2004), pp. 329–345. doi: [10.1137/S0036144503429121](https://doi.org/10.1137/S0036144503429121) (cited on pages 136, 138).
- [177] Nico Schlömer, Antonio Cervone, G. D. McBain, tryfon-mw, Bin Wang, Nate, Filip Gokstorp, Ruben van Staden, toothstone, Jørgen Schartum Dokken, Andrew Micallef, Dominic Kempf, Juan Sanchez, Keurfon Luu, anzil, Matthias Bussonnier, Raul Ciria Aylagas, Fred Fu, ivanmultiwave, Nils Wagner, Siwei Chen, tayebzaidi, Tomislav Maric, Amine Aboufirass, and Yuan Feng. *nschloe/pygmsh: None*. Version 7.1.11. 2021. doi: [10.5281/zenodo.5196231](https://doi.org/10.5281/zenodo.5196231) (cited on page 136).
- [178] J.-F. Remacle, Jonathan Lambrechts, Bruno Seny, Emilie Marchandise, Amaury Johnen, and C. Geuzainet. ‘Blossom-Quad: A non-uniform quadrilateral mesh generator using a minimum-cost perfect-matching algorithm’. In: *International journal for numerical methods in engineering* 89.9 (2012), pp. 1102–1119 (cited on page 138).
- [179] J.-F. Remacle, François Henrotte, T. Carrier-Baudouin, Eric Béchet, E. Marchandise, Christophe Geuzaine, and Thibaud Mouton. ‘A frontal Delaunay quad mesh generator using the L_∞ norm’. In: *International Journal for Numerical Methods in Engineering* 94.5 (2013), pp. 494–512 (cited on page 138).
- [180] Sándor P. Fekete, Kan Huang, Joseph S. B. Mitchell, Ojas Parekh, and Cynthia A. Phillips. ‘Geometric Hitting Set for Segments of Few Orientations’. In: *Theory Comput. Syst.* 62.2 (2018), pp. 268–303. doi: [10.1007/s00224-016-9744-7](https://doi.org/10.1007/s00224-016-9744-7) (cited on pages 155, 158).

- [181] Yu Zheng. ‘Trajectory Data Mining: An Overview’. In: *ACM Trans. Intell. Syst. Technol.* 6.3 (2015), 29:1–29:41. doi: [10.1145/2743025](https://doi.org/10.1145/2743025) (cited on page 155).
- [182] Yu Zheng and Xiaofang Zhou, eds. *Computing with Spatial Trajectories*. Springer, 2011 (cited on page 155).
- [183] Marc Benkert, Joachim Gudmundsson, Florian Hübner, and Thomas Wolle. ‘Reporting flock patterns’. In: *Comput. Geom.* 41.3 (2008), pp. 111–125. doi: [10.1016/j.comgeo.2007.10.003](https://doi.org/10.1016/j.comgeo.2007.10.003) (cited on page 155).
- [184] Joachim Gudmundsson and Marc J. van Kreveld. ‘Computing longest duration flocks in trajectory data’. In: *14th ACM International Symposium on Geographic Information Systems, ACM-GIS 2006, November 10-11, 2006, Arlington, Virginia, USA, Proceedings*. Ed. by Rolf A. de By and Silvia Nittel. ACM, 2006, pp. 35–42. doi: [10.1145/1183471.1183479](https://doi.org/10.1145/1183471.1183479) (cited on page 155).
- [185] Patrick Laube, Matt Duckham, and Thomas Wolle. ‘Decentralized Movement Pattern Detection amongst Mobile Geosensor Nodes’. In: *Geographic Information Science, 5th International Conference, GIScience 2008, Park City, UT, USA, September 23-26, 2008. Proceedings*. Ed. by Thomas J. Cova, Harvey J. Miller, Kate Beard, Andrew U. Frank, and Michael F. Goodchild. Vol. 5266. Lecture Notes in Computer Science. Springer, 2008, pp. 199–216. doi: [10.1007/978-3-540-87473-7_13](https://doi.org/10.1007/978-3-540-87473-7_13) (cited on page 155).
- [186] Marcos R. Vieira, Petko Bakalov, and Vassilis J. Tsotras. ‘On-line discovery of flock patterns in spatio-temporal data’. In: *17th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS 2009, November 4-6, 2009, Seattle, Washington, USA, Proceedings*. Ed. by Divyakant Agrawal, Walid G. Aref, Chang-Tien Lu, Mohamed F. Mokbel, Peter Scheuermann, Cyrus Shahabi, and Ouri Wolfson. ACM, 2009, pp. 286–295. doi: [10.1145/1653771.1653812](https://doi.org/10.1145/1653771.1653812) (cited on page 155).
- [187] Joachim Gudmundsson, Marc J. van Kreveld, and Bettina Speckmann. ‘Efficient Detection of Patterns in 2D Trajectories of Moving Points’. In: *GeoInformatica* 11.2 (2007), pp. 195–215. doi: [10.1007/s10707-006-0002-z](https://doi.org/10.1007/s10707-006-0002-z) (cited on pages 155, 156).
- [188] Mattias Andersson, Joachim Gudmundsson, Patrick Laube, and Thomas Wolle. ‘Reporting Leaders and Followers among Trajectories of Moving Point Objects’. In: *GeoInformatica* 12.4 (2008), pp. 497–528. doi: [10.1007/s10707-007-0037-9](https://doi.org/10.1007/s10707-007-0037-9) (cited on page 155).
- [189] Kevin Buchin, Maike Buchin, Marc J. van Kreveld, Bettina Speckmann, and Frank Staals. ‘Trajectory grouping structure’. In: *J. Comput. Geom.* 6.1 (2015), pp. 75–98. doi: [10.20382/jocg.v6i1a3](https://doi.org/10.20382/jocg.v6i1a3) (cited on page 156).
- [190] Yifan Li, Jiawei Han, and Jiong Yang. ‘Clustering moving objects’. In: *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*. Ed. by Won Kim, Ron Kohavi, Johannes Gehrke, and William DuMouchel. ACM, 2004, pp. 617–622. doi: [10.1145/1014052.1014129](https://doi.org/10.1145/1014052.1014129) (cited on page 156).
- [191] Marios Hadjieleftheriou, George Kollios, Dimitrios Gunopulos, and Vassilis J. Tsotras. ‘On-Line Discovery of Dense Areas in Spatio-temporal Databases’. In: *Advances in Spatial and Temporal Databases, 8th International Symposium, SSTD 2003, Santorini Island, Greece, July 24-27, 2003, Proceedings*. Ed. by Thanasis Hadzilacos, Yannis Manolopoulos, John F. Roddick, and Yannis Theodoridis. Vol. 2750. Lecture Notes in Computer Science. Springer, 2003, pp. 306–324. doi: [10.1007/978-3-540-45072-6_18](https://doi.org/10.1007/978-3-540-45072-6_18) (cited on page 156).
- [192] Jae-Gil Lee, Jiawei Han, Xiaolei Li, and Hector Gonzalez. ‘TraClass: trajectory classification using hierarchical region-based and trajectory-based clustering’. In: *Proc. VLDB Endow.* 1.1 (2008), pp. 1081–1094. doi: [10.14778/1453856.1453972](https://doi.org/10.14778/1453856.1453972) (cited on page 156).
- [193] Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. ‘Trajectory clustering: a partition-and-group framework’. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*. Ed. by Chee Yong Chan, Beng Chin Ooi, and Aoying Zhou. ACM, 2007, pp. 593–604. doi: [10.1145/1247480.1247546](https://doi.org/10.1145/1247480.1247546) (cited on page 156).
- [194] Mirco Nanni and Dino Pedreschi. ‘Time-focused clustering of trajectories of moving objects’. In: *J. Intell. Inf. Syst.* 27.3 (2006), pp. 267–289. doi: [10.1007/s10844-006-9953-7](https://doi.org/10.1007/s10844-006-9953-7) (cited on page 156).

- [195] Zhenhui Li, Bolin Ding, Jiawei Han, and Roland Kays. ‘Swarm: Mining Relaxed Temporal Moving Object Clusters’. In: *Proc. VLDB Endow.* 3.1 (2010), pp. 723–734. doi: [10.14778/1920841.1920934](https://doi.org/10.14778/1920841.1920934) (cited on page 156).
- [196] Muhammad Reaz Uddin, Chinya V. Ravishankar, and Vassilis J. Tsotras. ‘Finding Regions of Interest from Trajectory Data’. In: *12th IEEE International Conference on Mobile Data Management, MDM 2011, Luleå, Sweden, June 6-9, 2011, Volume 1*. Ed. by Arkady B. Zaslavsky, Panos K. Chrysanthos, Dik Lun Lee, Dipanjan Chakraborty, Vana Kalogeraki, Mohamed F. Mokbel, and Chi-Yin Chow. IEEE Computer Society, 2011, pp. 39–48. doi: [10.1109/MDM.2011.12](https://doi.org/10.1109/MDM.2011.12) (cited on page 156).
- [197] Pasin Manurangsi. ‘Almost-polynomial ratio ETH-hardness of approximating densest k-subgraph’. In: *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*. Ed. by Hamed Hatami, Pierre McKenzie, and Valerie King. ACM, 2017, pp. 954–961. doi: [10.1145/3055399.3055412](https://doi.org/10.1145/3055399.3055412) (cited on page 157).
- [198] Gerhard Reinelt. ‘TSPLIB - A Traveling Salesman Problem Library’. In: *INFORMS J. Comput.* 3.4 (1991), pp. 376–384. doi: [10.1287/ijoc.3.4.376](https://doi.org/10.1287/ijoc.3.4.376) (cited on pages 165, 225, 245).
- [199] Michal Piorowski, Natasa Sarafijanovic-Djukic, and Matthias Grossglauser. *CRAWDDAD dataset epfl/mobility (v. 2009-02-24)*, doi:10.15783/C7J010. 2009 (cited on page 170).
- [200] Erik Winfree. ‘Algorithmic self-assembly of DNA’. PhD thesis. California Institute of Technology, 1998 (cited on pages 177, 179).
- [201] Aaron T. Becker, Erik D. Demaine, Sándor P. Fekete, Golnaz Habibi, and James McLurkin. ‘Reconfiguring Massive Particle Swarms with Limited, Global Control’. In: *Algorithms for Sensor Systems - 9th International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics, ALGOSENSORS 2013, Sophia Antipolis, France, September 5-6, 2013, Revised Selected Papers*. Ed. by Paola Flocchini, Jie Gao, Evangelos Kranakis, and Friedhelm Meyer auf der Heide. Vol. 8243. Lecture Notes in Computer Science. Springer, 2013, pp. 51–66. doi: [10.1007/978-3-642-45346-5_5](https://doi.org/10.1007/978-3-642-45346-5_5) (cited on pages 177, 180, 205).
- [202] Sheryl Manzoor, Samuel Sheckman, Jarrett Lonsford, Hoyeon Kim, Min Jun Kim, and Aaron T. Becker. ‘Parallel Self-Assembly of Polyominoes Under Uniform Control Inputs’. In: *IEEE Robotics Autom. Lett.* 2.4 (2017), pp. 2040–2047. doi: [10.1109/LRA.2017.2715402](https://doi.org/10.1109/LRA.2017.2715402) (cited on pages 178, 179, 185).
- [203] Paul W. K. Rothmund and Erik Winfree. ‘The program-size complexity of self-assembled squares (extended abstract)’. In: *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*. Ed. by F. Frances Yao and Eugene M. Luks. ACM, 2000, pp. 459–468. doi: [10.1145/335305.335358](https://doi.org/10.1145/335305.335358) (cited on page 179).
- [204] Leonard M. Adleman, Qi Cheng, Ashish Goel, and Ming-Deh A. Huang. ‘Running time and program size for self-assembled squares’. In: *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*. Ed. by Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis. ACM, 2001, pp. 740–748. doi: [10.1145/380752.380881](https://doi.org/10.1145/380752.380881) (cited on page 179).
- [205] Sarah Cannon, Erik D. Demaine, Martin L. Demaine, Sarah Eisenstat, Matthew J. Patitz, Robert T. Schweller, Scott M. Summers, and Andrew Winslow. ‘Two Hands Are Better Than One (up to constant factors): Self-Assembly In The 2HAM vs. aTAM’. In: *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, February 27 - March 2, 2013, Kiel, Germany*. Ed. by Natacha Portier and Thomas Wilke. Vol. 20. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013, pp. 172–184. doi: [10.4230/LIPIcs.STACS.2013.172](https://doi.org/10.4230/LIPIcs.STACS.2013.172) (cited on page 179).
- [206] Ho-Lin Chen and David Doty. ‘Parallelism and Time in Hierarchical Self-Assembly’. In: *SIAM J. Comput.* 46.2 (2017), pp. 661–709. doi: [10.1137/151004161](https://doi.org/10.1137/151004161) (cited on page 179).
- [207] Erik D. Demaine, Martin L. Demaine, Sándor P. Fekete, Mashhood Ishaque, Eynat Rafalin, Robert T. Schweller, and Diane L. Souvaine. ‘Staged self-assembly: nanomanufacture of arbitrary shapes with $O(1)$ glues’. In: *Nat. Comput.* 7.3 (2008), pp. 347–370. doi: [10.1007/s11047-008-9073-0](https://doi.org/10.1007/s11047-008-9073-0) (cited on pages 180, 201).

- [208] Cameron T. Chalk, Eric Martinez, Robert T. Schweller, Luis Vega, Andrew Winslow, and Tim Wylie. ‘Optimal Staged Self-Assembly of General Shapes’. In: *Algorithmica* 80.4 (2018), pp. 1383–1409. doi: [10.1007/s00453-017-0318-0](https://doi.org/10.1007/s00453-017-0318-0) (cited on pages 180, 201).
- [209] Erik D. Demaine, Sándor P. Fekete, Christian Scheffer, and Arne Schmidt. ‘New geometric algorithms for fully connected staged self-assembly’. In: *Theor. Comput. Sci.* 671 (2017), pp. 4–18. doi: [10.1016/j.tcs.2016.11.020](https://doi.org/10.1016/j.tcs.2016.11.020) (cited on pages 180, 201).
- [210] Paul Seung Soo Kim, Aaron T. Becker, Yan Ou, Anak Agung Julius, and Min Jun Kim. ‘Imparting magnetic dipole heterogeneity to internalized iron oxide nanoparticles for microorganism swarm control’. In: *Journal of Nanoparticle Research* 17.3 (2015), pp. 1–15 (cited on page 180).
- [211] Paul Seung Soo Kim, Aaron T. Becker, Yan Ou, Anak Agung Julius, and MinJun Kim. ‘Swarm control of cell-based microrobots using a single global magnetic field’. In: *10th International Conference on Ubiquitous Robots and Ambient Intelligence, URAI 2013, Jeju, Korea (South), October 30 - Nov. 2, 2013*. IEEE, 2013, pp. 21–26. doi: [10.1109/URAI.2013.6677461](https://doi.org/10.1109/URAI.2013.6677461) (cited on page 180).
- [212] Aaron T. Becker, Ouajdi Felfoul, and Pierre E. Dupont. ‘Simultaneously powering and controlling many actuators with a clinical MRI scanner’. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, IL, USA, September 14-18, 2014*. IEEE, 2014, pp. 2017–2023. doi: [10.1109/IROS.2014.6942831](https://doi.org/10.1109/IROS.2014.6942831) (cited on page 180).
- [213] Aaron T. Becker, Ouajdi Felfoul, and Pierre E. Dupont. ‘Toward tissue penetration by MRI-powered millirobots using a self-assembled Gauss gun’. In: *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*. IEEE, 2015, pp. 1184–1189. doi: [10.1109/ICRA.2015.7139341](https://doi.org/10.1109/ICRA.2015.7139341) (cited on page 180).
- [214] Aaron T. Becker, Erik D. Demaine, Sándor P. Fekete, and James McLurkin. ‘Particle computation: Designing worlds to control robot swarms with only global signals’. In: *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014*. IEEE, 2014, pp. 6751–6756. doi: [10.1109/ICRA.2014.6907856](https://doi.org/10.1109/ICRA.2014.6907856) (cited on pages 180, 205).
- [215] Hamed Mohtasham Shad, Rose Morris-Wright, Erik D. Demaine, Sándor P. Fekete, and Aaron T. Becker. ‘Particle computation: Device fan-out and binary memory’. In: *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*. IEEE, 2015, pp. 5384–5389. doi: [10.1109/ICRA.2015.7139951](https://doi.org/10.1109/ICRA.2015.7139951) (cited on page 180).
- [216] Aaron T. Becker, Golnaz Habibi, Justin Werfel, Michael Rubenstein, and James McLurkin. ‘Massive uniform manipulation: Controlling large populations of simple robots with a common input signal’. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Tokyo, Japan, November 3-7, 2013*. IEEE, 2013, pp. 520–527. doi: [10.1109/IROS.2013.6696401](https://doi.org/10.1109/IROS.2013.6696401) (cited on page 180).
- [217] Aaron T. Becker, Chris Ertel, and James McLurkin. ‘Crowdsourcing swarm manipulation experiments: A massive online user study with large swarms of simple robots’. In: *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014*. IEEE, 2014, pp. 2825–2830. doi: [10.1109/ICRA.2014.6907264](https://doi.org/10.1109/ICRA.2014.6907264) (cited on page 180).
- [218] Shiva Shahrokhi and Aaron T. Becker. ‘Stochastic swarm control with global inputs’. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015, Hamburg, Germany, September 28 - October 2, 2015*. IEEE, 2015, pp. 421–427. doi: [10.1109/IROS.2015.7353407](https://doi.org/10.1109/IROS.2015.7353407) (cited on page 180).
- [219] Justin Werfel and Radhika Nagpal. ‘Extended Stigmergy in Collective Construction’. In: *IEEE Intell. Syst.* 21.2 (2006), pp. 20–28. doi: [10.1109/MIS.2006.25](https://doi.org/10.1109/MIS.2006.25) (cited on page 180).
- [220] Justin Werfel and Radhika Nagpal. ‘Three-Dimensional Construction with Mobile Robots and Modular Blocks’. In: *Int. J. Robotics Res.* 27.3-4 (2008), pp. 463–479. doi: [10.1177/0278364907084984](https://doi.org/10.1177/0278364907084984) (cited on page 180).

- [221] Zahra Derakhshandeh, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. ‘An Algorithmic Framework for Shape Formation Problems in Self-Organizing Particle Systems’. In: *Proceedings of the Second Annual International Conference on Nanoscale Computing and Communication, NANOCOM’ 15, Boston, MA, USA, September 21-22, 2015*. Ed. by Faramarz Fekri, Sasitharan Balasubramaniam, Tommaso Melodia, Ahmad Beirami, and Albert Cabellos. ACM, 2015, 21:1–21:2. doi: [10.1145/2800795.2800829](https://doi.org/10.1145/2800795.2800829) (cited on pages 180, 263).
- [222] Zahra Derakhshandeh, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. ‘Universal coating for programmable matter’. In: *Theor. Comput. Sci.* 671 (2017), pp. 56–68. doi: [10.1016/j.tcs.2016.02.039](https://doi.org/10.1016/j.tcs.2016.02.039) (cited on page 180).
- [223] Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Christian Scheffer, and Henk Meijer. ‘Coordinated Motion Planning: Reconfiguring a Swarm of Labeled Robots with Bounded Stretch’. In: *34th International Symposium on Computational Geometry, SoCG 2018, June 11-14, 2018, Budapest, Hungary*. Ed. by Bettina Speckmann and Csaba D. Tóth. Vol. 99. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 29:1–29:15. doi: [10.4230/LIPIcs.SocG.2018.29](https://doi.org/10.4230/LIPIcs.SocG.2018.29) (cited on pages 180, 263, 264).
- [224] Aaron T. Becker, Sándor P. Fekete, Phillip Keldenich, Matthias Konitzny, Lillian Lin, and Christian Scheffer. ‘Coordinated Motion Planning: The Video’. In: *Symposium on Computational Geometry (SoCG)*. Video at <https://www.ibr.cs.tu-bs.de/users/fekete/Videos/CoordinatedMotionPlanning.mp4>. 2018, 74:1–74:6. doi: [10.4230/LIPIcs.SocG.2018.74](https://doi.org/10.4230/LIPIcs.SocG.2018.74) (cited on pages 180, 264).
- [225] Dan Arbuckle and Aristides A. G. Requicha. ‘Self-assembly and self-repair of arbitrary shapes by a swarm of reactive robots: algorithms and simulations’. In: *Auton. Robots* 28.2 (2010), pp. 197–211. doi: [10.1007/s10514-009-9162-7](https://doi.org/10.1007/s10514-009-9162-7) (cited on page 180).
- [226] Anupama J. Thubagere, Wei Li, Robert F. Johnson, Zibo Chen, Shayan Doroudi, Yae Lim Lee, Gregory Izatt, Sarah Wittman, Niranjan Srinivas, Damien Woods, Erik Winfree, and Lulu Qian. ‘A cargo-sorting DNA robot’. In: *Science* 357.6356 (2017), eaan6558 (cited on page 180).
- [227] Michael Rubenstein, Adrian Cabrera, Justin Werfel, Golnaz Habibi, James McLurkin, and Radhika Nagpal. ‘Collective transport of complex objects by simple robots: theory and experiments’. In: *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS ’13, Saint Paul, MN, USA, May 6-10, 2013*. Ed. by Maria L. Gini, Onn Shehory, Takayuki Ito, and Catholijn M. Jonker. IFAAMAS, 2013, pp. 47–54 (cited on page 180).
- [228] Michael Hoffmann. ‘Push-* is NP-hard’. In: *Proceedings of the 12th Canadian Conference on Computational Geometry, Fredericton, New Brunswick, Canada, August 16-19, 2000*. 2000 (cited on page 180).
- [229] Piotr Berman and Georg Schnitger. ‘On the Complexity of Approximating the Independent Set Problem’. In: *Inf. Comput.* 96.1 (1992), pp. 77–94. doi: [10.1016/0890-5401\(92\)90056-L](https://doi.org/10.1016/0890-5401(92)90056-L) (cited on page 191).
- [230] Alexey Ignatiev, António Morgado, and João Marques-Silva. ‘PySAT: A Python Toolkit for Prototyping with SAT Oracles’. In: *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*. Ed. by Olaf Beyersdorff and Christoph M. Wintersteiger. Vol. 10929. Lecture Notes in Computer Science. Springer, 2018, pp. 428–437. doi: [10.1007/978-3-319-94144-8_26](https://doi.org/10.1007/978-3-319-94144-8_26) (cited on page 199).
- [231] Gilles Audemard and Laurent Simon. ‘On the Glucose SAT Solver’. In: *Int. J. Artif. Intell. Tools* 27.1 (2018), 1840001:1–1840001:25. doi: [10.1142/S0218213018400018](https://doi.org/10.1142/S0218213018400018) (cited on page 199).
- [232] Jia Hui Liang, Chanseok Oh, Vijay Ganesh, Krzysztof Czarnecki, and Pascal Poupart. ‘Maple-COMSPS, MapleCOMSPS LRB, MapleCOMSPS CHB’. In: *Proceedings of SAT Competition 2016* (2016) (cited on page 199).
- [233] Norbert Manthey. ‘The MergeSat Solver’. In: *Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings*. Ed. by Chu-Min Li and Felip Manyà. Vol. 12831. Lecture Notes in Computer Science. Springer, 2021, pp. 387–398. doi: [10.1007/978-3-030-80223-3_27](https://doi.org/10.1007/978-3-030-80223-3_27) (cited on page 199).

- [234] Niklas Eén and Niklas Sörensson. 'An Extensible SAT-solver'. In: *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*. Ed. by Enrico Giunchiglia and Armando Tacchella. Vol. 2919. Lecture Notes in Computer Science. Springer, 2003, pp. 502–518. doi: [10.1007/978-3-540-24605-3_37](https://doi.org/10.1007/978-3-540-24605-3_37) (cited on page 199).
- [235] Arne Schmidt, Sheryl Manzoor, Li Huang, Aaron T. Becker, and Sándor P. Fekete. 'Efficient Parallel Self-Assembly Under Uniform Control Inputs'. In: *IEEE Robotics Autom. Lett.* 3.4 (2018), pp. 3521–3528. doi: [10.1109/LRA.2018.2853758](https://doi.org/10.1109/LRA.2018.2853758) (cited on page 201).
- [236] Pierre Pouponneau, Jean-Christophe Leroux, and Sylvain Martel. 'Magnetic nanoparticles encapsulated into biodegradable microparticles steered with an upgraded magnetic resonance imaging system for tumor chemoembolization'. In: *Biomaterials* 30.31 (2009), pp. 6327–6332 (cited on pages 204, 205).
- [237] Julia Litvinov, Azeem Nasrullah, Timothy Sherlock, Yi-Ju Wang, Paul Ruchhoeft, and Richard C. Willson. 'High-throughput top-down fabrication of uniform magnetic particles'. In: *PloS one* 7.5 (2012), e37440 (cited on page 204).
- [238] Lyès Mellal, David Folio, Karim Belharet, and Antoine Ferreira. 'Magnetic microbot design framework for antiangiogenic tumor therapy'. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015, Hamburg, Germany, September 28 - October 2, 2015*. IEEE, 2015, pp. 1397–1402. doi: [10.1109/IROS.2015.7353550](https://doi.org/10.1109/IROS.2015.7353550) (cited on pages 204, 205).
- [239] Jean-Baptiste Mathieu and Sylvain Martel. 'Magnetic microparticle steering within the constraints of an MRI system: proof of concept of a novel targeting approach'. In: *Biomedical microdevices* 9.6 (2007), pp. 801–808 (cited on page 205).
- [240] Aaron T. Becker, Erik D. Demaine, Sándor P. Fekete, Jarrett Lonsford, and Rose Morris-Wright. 'Particle computation: complexity, algorithms, and logic'. In: *Nat. Comput.* 18.1 (2019), pp. 181–201. doi: [10.1007/s11047-017-9666-6](https://doi.org/10.1007/s11047-017-9666-6) (cited on page 205).
- [241] Aaron T. Becker, Erik D. Demaine, Sándor P. Fekete, Hamed Mohtasham Shad, and Rose Morris-Wright. 'Tilt: The Video - Designing Worlds to Control Robot Swarms with Only Global Signals'. In: *31st International Symposium on Computational Geometry, SoCG 2015, June 22-25, 2015, Eindhoven, The Netherlands*. Ed. by Lars Arge and János Pach. Vol. 34. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015, pp. 16–18. doi: [10.4230/LIPIcs.SOCG.2015.16](https://doi.org/10.4230/LIPIcs.SOCG.2015.16) (cited on page 205).
- [242] Sagar Chowdhury, Wuming Jing, and David J. Cappelleri. 'Controlling multiple microrobots: recent progress and future challenges'. In: *Journal of Micro-Bio Robotics* 10.1-4 (2015), pp. 1–11. doi: [10.1007/s12213-015-0083-6](https://doi.org/10.1007/s12213-015-0083-6) (cited on page 205).
- [243] Jose Balanza-Martinez, Austin Luchsinger, David Caballero, Rene Reyes, Angel A. Cantu, Robert T. Schweller, Luis Angel Garcia, and Tim Wylie. 'Full Tilt: Universal Constructors for General Shapes with Uniform External Forces'. In: *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*. Ed. by Timothy M. Chan. SIAM, 2019, pp. 2689–2708. doi: [10.1137/1.9781611975482.167](https://doi.org/10.1137/1.9781611975482.167) (cited on page 205).
- [244] Yinan Zhang, Xiaolei Chen, Hang Qi, and Devin J. Balkcom. 'Rearranging agents in a small space using global controls'. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017*. IEEE, 2017, pp. 3576–3582. doi: [10.1109/IROS.2017.8206202](https://doi.org/10.1109/IROS.2017.8206202) (cited on page 205).
- [245] Yinan Zhang, Emily Whiting, and Devin J. Balkcom. 'Assembling and Disassembling Planar Structures With Divisible and Atomic Components'. In: *IEEE Trans Autom. Sci. Eng.* 15.3 (2018), pp. 945–954. doi: [10.1109/TASE.2018.2809595](https://doi.org/10.1109/TASE.2018.2809595) (cited on page 205).
- [246] Steve Alpern and Shmuel Gal. *The theory of search games and rendezvous*. Vol. 55. International series in operations research and management science. Kluwer, 2003 (cited on page 205).
- [247] Edward J. Anderson and Sándor P. Fekete. 'Two Dimensional Rendezvous Search'. In: *Oper. Res.* 49.1 (2001), pp. 107–118. doi: [10.1287/opre.49.1.107](https://doi.org/10.1287/opre.49.1.107). doi: [10.11191](https://doi.org/10.11191) (cited on page 205).

- [248] Malika Meghjani and Gregory Dudek. ‘Multi-robot exploration and rendezvous on graphs’. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*. IEEE, 2012, pp. 5270–5276. doi: [10.1109/IROS.2012.6386049](https://doi.org/10.1109/IROS.2012.6386049) (cited on page 205).
- [249] Paola Flocchini. ‘Gathering’. In: *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*. 2019, pp. 63–82 (cited on page 205).
- [250] Richard M. Karp. ‘Reducibility Among Combinatorial Problems’. In: *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*. Ed. by Raymond E. Miller and James W. Thatcher. The IBM Research Symposia Series. Plenum Press, New York, 1972, pp. 85–103. doi: [10.1007/978-1-4684-2001-2_9](https://doi.org/10.1007/978-1-4684-2001-2_9) (cited on page 206).
- [251] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. ‘Playing Atari with Deep Reinforcement Learning’. In: *CoRR abs/1312.5602* (2013) (cited on page 212).
- [252] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharmashan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. ‘Human-level control through deep reinforcement learning’. In: *Nat.* 518.7540 (2015), pp. 529–533. doi: [10.1038/nature14236](https://doi.org/10.1038/nature14236) (cited on page 212).
- [253] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. *Stable Baselines*. <https://github.com/hill-a/stable-baselines>. 2018 (cited on pages 212, 217).
- [254] Ronald J. Williams. ‘Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning’. In: *Mach. Learn.* 8 (1992), pp. 229–256. doi: [10.1007/BF00992696](https://doi.org/10.1007/BF00992696) (cited on page 213).
- [255] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. ‘Proximal Policy Optimization Algorithms’. In: *CoRR abs/1707.06347* (2017) (cited on page 213).
- [256] Laura Graesser and Wah Loon Keng. *Foundations of deep reinforcement learning: theory and practice in Python*. Addison-Wesley Professional, 2019 (cited on page 214).
- [257] Matthias Konitzny. ‘Reinforcement Learning for Navigating Particle Swarms by Global Force’. Master’s thesis. TU Braunschweig, Institute of Operating Systems and Computer Networks, Algorithms Division, 2019 (cited on page 219).
- [258] Ambros M. Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp Christophel, Kati Jarck, Thorsten Koch, Jeff T. Linderoth, Marco E. Lübbecke, Hans D. Mittelmann, Derya B. Özyurt, Ted K. Ralphs, Domenico Salvagnin, and Yuji Shinano. ‘MIPLIB 2017: data-driven compilation of the 6th mixed-integer programming library’. In: *Math. Program. Comput.* 13.3 (2021), pp. 443–490. doi: [10.1007/s12532-020-00194-3](https://doi.org/10.1007/s12532-020-00194-3) (cited on page 225).
- [259] Wolfgang Mulzer and Günter Rote. ‘Minimum-weight triangulation is NP-hard’. In: *J. ACM* 55.2 (2008), 11:1–11:29. doi: [10.1145/1346330.1346336](https://doi.org/10.1145/1346330.1346336) (cited on page 237).
- [260] Andreas Haas. ‘Solving Large-Scale Minimum-Weight Triangulation Instances to Provable Optimality’. In: *34th International Symposium on Computational Geometry, SoCG 2018, June 11-14, 2018, Budapest, Hungary*. Ed. by Bettina Speckmann and Csaba D. Tóth. Vol. 99. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 44:1–44:14. doi: [10.4230/LIPIcs.SoCG.2018.44](https://doi.org/10.4230/LIPIcs.SoCG.2018.44) (cited on page 237).
- [261] Kate Amanda Smith-Miles. ‘Towards insightful algorithm selection for optimisation using meta-learning concepts’. In: *Proceedings of the International Joint Conference on Neural Networks, IJCNN 2008, part of the IEEE World Congress on Computational Intelligence, WCCI 2008, Hong Kong, China, June 1-6, 2008*. IEEE, 2008, pp. 4118–4124. doi: [10.1109/IJCNN.2008.4634391](https://doi.org/10.1109/IJCNN.2008.4634391) (cited on page 241).

- [262] Kate Smith-Miles, Jano I. van Hemert, and Xin Yu Lim. ‘Understanding TSP Difficulty by Learning from Evolved Instances’. In: *Learning and Intelligent Optimization, 4th International Conference, LION 4, Venice, Italy, January 18-22, 2010. Selected Papers*. Ed. by Christian Blum and Roberto Battiti. Vol. 6073. Lecture Notes in Computer Science. Springer, 2010, pp. 266–280. doi: [10.1007/978-3-642-13800-3_29](https://doi.org/10.1007/978-3-642-13800-3_29) (cited on page 241).
- [263] Kate Smith-Miles and Jano I. van Hemert. ‘Discovering the suitability of optimisation algorithms by learning from evolved instances’. In: *Ann. Math. Artif. Intell.* 61.2 (2011), pp. 87–104. doi: [10.1007/s10472-011-9230-5](https://doi.org/10.1007/s10472-011-9230-5) (cited on page 241).
- [264] Kate Smith-Miles and Thomas T. Tan. ‘Measuring algorithm footprints in instance space’. In: *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2012, Brisbane, Australia, June 10-15, 2012*. IEEE, 2012, pp. 1–8. doi: [10.1109/CEC.2012.6252992](https://doi.org/10.1109/CEC.2012.6252992) (cited on page 241).
- [265] Kate Smith-Miles and Leo Lopes. ‘Measuring instance difficulty for combinatorial optimization problems’. In: *Comput. Oper. Res.* 39.5 (2012), pp. 875–889. doi: [10.1016/j.cor.2011.07.006](https://doi.org/10.1016/j.cor.2011.07.006) (cited on page 241).
- [266] Kate Smith-Miles, Davaatseren Baatar, Brendan Wreford, and Rhyd Lewis. ‘Towards objective measures of algorithm performance across instance space’. In: *Comput. Oper. Res.* 45 (2014), pp. 12–24. doi: [10.1016/j.cor.2013.11.015](https://doi.org/10.1016/j.cor.2013.11.015) (cited on page 241).
- [267] Kate Smith-Miles and Simon Bowly. ‘Generating new test instances by evolving in instance space’. In: *Comput. Oper. Res.* 63 (2015), pp. 102–113. doi: [10.1016/j.cor.2015.04.022](https://doi.org/10.1016/j.cor.2015.04.022) (cited on page 241).
- [268] John R. Rice. ‘The algorithm selection problem’. In: *Advances in computers*. Vol. 15. Elsevier, 1976, pp. 65–118 (cited on page 241).
- [269] *Melbourne Algorithm Test Instance Library with Data Analytics (MATILDA)*. URL: <https://matilda.unimelb.edu.au/matilda/> (cited on page 241).
- [270] Joseph O’Rourke. *Polyhedral object models from 3D points*. Tech. rep. IFI-HH-M-77/80. Universität Hamburg, 1980 (cited on pages 243, 244).
- [271] Sándor P. Fekete. ‘Geometry and the Travelling Salesman Problem’. Ph.D. Thesis. Waterloo, ON: Department of Combinatorics and Optimization, University of Waterloo, 1992 (cited on pages 244, 245, 247, 248).
- [272] Sándor P. Fekete and William R. Pulleyblank. ‘Area Optimization of Simple Polygons’. In: *Proceedings of the Ninth Annual Symposium on Computational Geometry San Diego, CA, USA, May 19-21, 1993*. Ed. by Chee Yap. ACM, 1993, pp. 173–182. doi: [10.1145/160985.161016](https://doi.org/10.1145/160985.161016) (cited on pages 244, 245).
- [273] Sándor P. Fekete. ‘On simple polygonizations with optimal area’. In: *Discrete & Computational Geometry* 23.1 (2000), pp. 73–110 (cited on pages 244, 245, 247).
- [274] Joseph S. B. Mitchell. *Approximation algorithms for geometric separation problems*. Tech. rep. SUNY Stony Brook, 1993 (cited on page 244).
- [275] Joseph S. B. Mitchell and Subhash Suri. ‘Separation and Approximation of Polyhedral Objects’. In: *Comput. Geom.* 5 (1995), pp. 95–114. doi: [10.1016/0925-7721\(95\)00006-U](https://doi.org/10.1016/0925-7721(95)00006-U) (cited on page 244).
- [276] Oswin Aichholzer, Franz Aurenhammer, and Hannes Krasser. ‘Enumerating Order Types for Small Point Sets with Applications’. In: *Order* 19.3 (2002), pp. 265–281. doi: [10.1023/A:1021231927255](https://doi.org/10.1023/A:1021231927255) (cited on page 244).
- [277] Chong Zhu, Gopalakrishnan Sundaram, Jack Snoeyink, and Joseph S. B. Mitchell. ‘Generating Random Polygons with Given Vertices’. In: *Comput. Geom.* 6 (1996), pp. 277–290. doi: [10.1016/0925-7721\(95\)00031-3](https://doi.org/10.1016/0925-7721(95)00031-3) (cited on page 244).
- [278] Eyal Ackerman, Oswin Aichholzer, and Balázs Keszegh. ‘Improved upper bounds on the reflexivity of point sets’. In: *Comput. Geom.* 42.3 (2009), pp. 241–249. doi: [10.1016/j.comgeo.2008.05.004](https://doi.org/10.1016/j.comgeo.2008.05.004) (cited on page 244).
- [279] Manuel Abellanas, Jesus Garcia-Lopez, Gregorio Hernández-Peñalver, Ferran Hurtado, Oriol Serra, and Jorge Urrutia. ‘Updating Polygonizations’. In: *Comput. Graph. Forum* 12.3 (1993), pp. 143–152. doi: [10.1111/1467-8659.1230143](https://doi.org/10.1111/1467-8659.1230143) (cited on page 244).

- [280] Esther M. Arkin, Joseph S. B. Mitchell, Sándor P. Fekete, Ferran Hurtado, Marc Noy, Vera Sacristán, and Saurabh Sethia. ‘On the reflexivity of point sets’. In: *Discrete and Computational Geometry*. Springer, 2003, pp. 139–156 (cited on page 244).
- [281] Thomas Auer and Martin Held. ‘Heuristics for the Generation of Random Polygons’. In: *Proceedings of the 8th Canadian Conference on Computational Geometry, Carleton University, Ottawa, Canada, August 12-15, 1996*. Ed. by Frank Fiala, Evangelos Kranakis, and Jörg-Rüdiger Sack. Carleton University Press, 1996, pp. 38–43 (cited on page 244).
- [282] Joseph O’Rourke, Subhash Suri, and Csaba D. Tóth. ‘Polygons’. In: *Handbook of discrete and computational geometry*. Ed. by Csaba D. Toth, Joseph O’Rourke, and Jacob E. Goodman. CRC Press, 2017. Chap. 30, pp. 787–810 (cited on page 244).
- [283] Alfredo García Olaverri, Marc Noy, and Javier Tejel. ‘Lower bounds on the number of crossing-free subgraphs of K_N ’. In: *Comput. Geom.* 16.4 (2000), pp. 211–221. doi: [10.1016/S0925-7721\(00\)00010-9](https://doi.org/10.1016/S0925-7721(00)00010-9) (cited on page 244).
- [284] Micha Sharir, Adam Sheffer, and Emo Welzl. ‘Counting plane graphs: Perfect matchings, spanning cycles, and Kasteleyn’s technique’. In: *J. Comb. Theory, Ser. A* 120.4 (2013), pp. 777–794. doi: [10.1016/j.jcta.2013.01.002](https://doi.org/10.1016/j.jcta.2013.01.002) (cited on page 244).
- [285] Maria Teresa Taranilla, Edilma Olinda Gagliardi, and Gregorio Hernández Peñalver. ‘Approaching minimum area polygonization’. In: *XVII Congreso Argentino de Ciencias de la Computacion*. 2011, pp. 31–40 (cited on page 245).
- [286] Jiju Peethambaran, Amal Dev Parakkat, and Ramanathan Muthuganapathy. ‘An Empirical Study on Randomized Optimal Area Polygonization of Planar Point Sets’. In: *ACM J. Exp. Algorithmics* 21.1 (2016), 1.10:1–1.10:24. doi: [10.1145/2896849](https://doi.org/10.1145/2896849) (cited on page 245).
- [287] Jiju Peethambaran, Amal Dev Parakkat, and Ramanathan Muthuganapathy. ‘A Randomized Approach to Volume Constrained Polyhedronization Problem’. In: *J. Comput. Inf. Sci. Eng.* 15.1 (2015). doi: [10.1115/1.4029559](https://doi.org/10.1115/1.4029559) (cited on page 245).
- [288] Julien Lepagnet, Laurent Moalic, and Dominique Schmitt. ‘Optimal area polygonization by triangulation and ray-tracing’. In: *Journal of Experimental Algorithmics* (2020). Submitted (cited on pages 245, 250).
- [289] Loïc Crombez, Guilherme D. da Fonseca, and Yan Gerard. ‘Greedy and Local Search Solutions to the Minimum and Maximum Area’. In: *Journal of Experimental Algorithmics* (2020). Submitted (cited on pages 245, 250).
- [290] Nir Goren, Efi Fogel, and Dan Halperin. ‘Area-optimal polygonization using simulated annealing’. In: *Journal of Experimental Algorithmics* (2020). Submitted (cited on pages 245, 250).
- [291] Günther Eder, Martin Held, Steinthor Jasonarson, Philipp Mayer, and Peter Palfrader. ‘2-Opt moves and flips for area-optimal polygonalizations’. In: *Journal of Experimental Algorithmics* (2020). Submitted (cited on pages 246, 250).
- [292] Natanael Ramos, Rai Caetan de Jesus, Pedro de Rezende, Cid de Souza, and Fabio Luiz Usberti. ‘Heuristics for area optimal polygonizations’. In: *Journal of Experimental Algorithmics* (2020). Submitted (cited on pages 246, 250).
- [293] Sándor P. Fekete, Andreas Haas, Phillip Keldenich, Michael Perk, and Arne Schmidt. ‘Computing area-optimal simple polygonalization’. In: *Journal of Experimental Algorithmics* (2020). Submitted (cited on pages 246, 250).
- [294] Joseph O’Rourke. *Computational geometry in C*. Cambridge university press, 1998 (cited on page 246).
- [295] Georg Pick. ‘Geometrisches zur Zahlenlehre’. In: *Sitzungsberichte des Deutschen Naturwissenschaftlich-Medicinischen Vereines für Böhmen “Lotos” in Prag* 19 (1899), pp. 311–319 (cited on page 246).
- [296] W. W. Funkenbusch. ‘From Euler’s formula to Pick’s formula using an edge theorem’. In: *The American Mathematical Monthly* 81.6 (1974), pp. 647–648 (cited on page 246).
- [297] Harold Scott Macdonald Coxeter. *Introduction to Geometry*. Wiley, New York, 1969 (cited on page 246).

- [298] R. W. Gaskell, M. S. Klamkin, and P. Watson. ‘Triangulations and Pick’s theorem’. In: *Mathematics Magazine* 49.1 (1976), pp. 35–37 (cited on page 246).
- [299] Ivan Niven and H. S. Zuckerman. ‘Lattice points and polygonal area’. In: *The American Mathematical Monthly* 74.10 (1967), pp. 1195–1200 (cited on page 246).
- [300] Ren Ding and John R. Reay. ‘The boundary characteristic and Pick’s theorem in the Archimedean planar tilings’. In: *J. Comb. Theory, Ser. A* 44.1 (1987), pp. 110–119. doi: [10.1016/0097-3165\(87\)90063-X](https://doi.org/10.1016/0097-3165(87)90063-X) (cited on page 247).
- [301] John E. Reeve. ‘On the volume of lattice polyhedra’. In: *Proceedings of the London Mathematical Society* 3.1 (1957), pp. 378–395 (cited on page 247).
- [302] Bernard Chazelle and David P. Dobkin. ‘Decomposing a Polygon into its Convex Parts’. In: *Proceedings of the 11th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA*. Ed. by Michael J. Fischer, Richard A. DeMillo, Nancy A. Lynch, Walter A. Burkhard, and Alfred V. Aho. ACM, 1979, pp. 38–48. doi: [10.1145/800135.804396](https://doi.org/10.1145/800135.804396) (cited on page 255).
- [303] D. H. Greene. ‘The decomposition of polygons in convex parts’. In: *Computational Geometry*. Ed. by F.P. Preparata. Vol. 1. Advances in Computing Research. JAI Press, 1983, pp. 235–259 (cited on page 255).
- [304] J. Mark Keil. ‘Decomposing a Polygon into Simpler Components’. In: *SIAM J. Comput.* 14.4 (1985), pp. 799–817. doi: [10.1137/0214056](https://doi.org/10.1137/0214056) (cited on page 255).
- [305] J. Mark Keil and Jack Snoeyink. ‘On the Time Bound for Convex Decomposition of Simple Polygons’. In: *Int. J. Comput. Geom. Appl.* 12.3 (2002), pp. 181–192. doi: [10.1142/S0218195902000803](https://doi.org/10.1142/S0218195902000803) (cited on page 256).
- [306] Nicolas Grelier. ‘Minimum Convex Partition of Point Sets is NP-Hard’. In: *CoRR* abs/1911.07697 (2019) (cited on pages 256, 259).
- [307] Thomas Fevens, Henk Meijer, and David Rappaport. ‘Minimum convex partition of a constrained point set’. In: *Discret. Appl. Math.* 109.1-2 (2001), pp. 95–107. doi: [10.1016/S0166-218X\(00\)00237-7](https://doi.org/10.1016/S0166-218X(00)00237-7) (cited on page 256).
- [308] Christian Knauer and Andreas Spillner. ‘Approximation Algorithms for the Minimum Convex Partition Problem’. In: *Algorithm Theory - SWAT 2006, 10th Scandinavian Workshop on Algorithm Theory, Riga, Latvia, July 6-8, 2006, Proceedings*. Ed. by Lars Arge and Rusins Freivalds. Vol. 4059. Lecture Notes in Computer Science. Springer, 2006, pp. 232–241. doi: [10.1007/11785293_23](https://doi.org/10.1007/11785293_23) (cited on page 256).
- [309] Jorge Urrutia. ‘Open Problem Session’. In: *Canadian Conference on Computational Geometry (CCCG)*. 1998 (cited on page 256).
- [310] Victor Neumann-Lara, Eduardo Rivera-Campo, and Jorge Urrutia. ‘A Note on Convex Decompositions of a Set of Points in the Plane’. In: *Graphs Comb.* 20.2 (2004), pp. 223–231. doi: [10.1007/s00373-004-0555-2](https://doi.org/10.1007/s00373-004-0555-2) (cited on page 256).
- [311] Mario Lomelí-Haro. ‘Minimal Convex Decompositions’. In: *CoRR* abs/1207.3468 (2012) (cited on page 256).
- [312] Kiyoshi Hosono. ‘On convex decompositions of a planar point set’. In: *Discret. Math.* 309.6 (2009), pp. 1714–1717. doi: [10.1016/j.disc.2008.02.008](https://doi.org/10.1016/j.disc.2008.02.008) (cited on page 256).
- [313] Toshinori Sakai and Jorge Urrutia. ‘Convex decompositions of point sets in the plane’. In: *arXiv* 1909.06105 (2019), pp. 1–9 (cited on page 256).
- [314] Jesús García-López and Carlos M. Nicolás. ‘Planar Point Sets With Large Minimum Convex Decompositions’. In: *Graphs Comb.* 29.5 (2013), pp. 1347–1353. doi: [10.1007/s00373-012-1181-z](https://doi.org/10.1007/s00373-012-1181-z) (cited on page 256).
- [315] Allan S. Barboza, Cid C. de Souza, and Pedro J. de Rezende. ‘Minimum Convex Partition of Point Sets’. In: *Algorithms and Complexity - 11th International Conference, CIAC 2019, Rome, Italy, May 27-29, 2019, Proceedings*. Ed. by Pinar Heggernes. Vol. 11485. Lecture Notes in Computer Science. Springer, 2019, pp. 25–37. doi: [10.1007/978-3-030-17402-6_3](https://doi.org/10.1007/978-3-030-17402-6_3) (cited on page 256).

- [316] Hadrien Cambazard and Nicolas Catusse. ‘An Integer Programming Formulation Using Convex Polygons for the Convex Partition Problem’. In: *37th International Symposium on Computational Geometry, SoCG 2021, June 7-11, 2021, Buffalo, NY, USA (Virtual Conference)*. Ed. by Kevin Buchin and Éric Colin de Verdière. Vol. 189. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 20:1–20:13. doi: [10.4230/LIPIcs.SoCG.2021.20](https://doi.org/10.4230/LIPIcs.SoCG.2021.20) (cited on page 256).
- [317] Da Wei Zheng, Jack Spalding-Jamieson, and Brandon Zhang. ‘Computing Low-Cost Convex Partitions for Planar Point Sets with Randomized Local Search and Constraint Programming (CG Challenge)’. In: *36th International Symposium on Computational Geometry, SoCG 2020, June 23-26, 2020, Zürich, Switzerland*. Ed. by Sergio Cabello and Danny Z. Chen. Vol. 164. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 83:1–83:7. doi: [10.4230/LIPIcs.SoCG.2020.83](https://doi.org/10.4230/LIPIcs.SoCG.2020.83) (cited on pages 258, 259).
- [318] Laurent Moalic, Dominique Schmitt, Julien Lepagnot, and Julien Kitter. ‘Computing Low-Cost Convex Partitions for Planar Point Sets Based on a Memetic Approach (CG Challenge)’. In: *36th International Symposium on Computational Geometry, SoCG 2020, June 23-26, 2020, Zürich, Switzerland*. Ed. by Sergio Cabello and Danny Z. Chen. Vol. 164. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 84:1–84:9. doi: [10.4230/LIPIcs.SoCG.2020.84](https://doi.org/10.4230/LIPIcs.SoCG.2020.84) (cited on page 259).
- [319] Günther Eder, Martin Held, Stefan de Lorenzo, and Peter Palfrader. ‘Computing Low-Cost Convex Partitions for Planar Point Sets Based on Tailored Decompositions (CG Challenge)’. In: *36th International Symposium on Computational Geometry, SoCG 2020, June 23-26, 2020, Zürich, Switzerland*. Ed. by Sergio Cabello and Danny Z. Chen. Vol. 164. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 85:1–85:11. doi: [10.4230/LIPIcs.SoCG.2020.85](https://doi.org/10.4230/LIPIcs.SoCG.2020.85) (cited on page 259).
- [320] Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. ‘Programmable self-assembly in a thousand-robot swarm’. In: *Science* 345.6198 (2014), pp. 795–799 (cited on page 262).
- [321] Erol Sahin and Alan F. T. Winfield. ‘Special issue on swarm robotics’. In: *Swarm Intell.* 2.2-4 (2008), pp. 69–72. doi: [10.1007/s11721-008-0020-6](https://doi.org/10.1007/s11721-008-0020-6) (cited on page 262).
- [322] Soon-Jo Chung, Aditya A. Paranjape, Philip M. Dames, Shaojie Shen, and Vijay Kumar. ‘A Survey on Aerial Swarm Robotics’. In: *IEEE Trans. Robotics* 34.4 (2018), pp. 837–855. doi: [10.1109/TR0.2018.2857475](https://doi.org/10.1109/TR0.2018.2857475) (cited on page 262).
- [323] Matthew Turpin, Kartik Mohta, Nathan Michael, and Vijay Kumar. ‘Goal assignment and trajectory planning for large teams of interchangeable robots’. In: *Auton. Robots* 37.4 (2014), pp. 401–415. doi: [10.1007/s10514-014-9412-1](https://doi.org/10.1007/s10514-014-9412-1) (cited on page 262).
- [324] Daniel Delahaye, Stéphane Puechmorel, Panagiotis Tsiotras, and Eric Féron. ‘Mathematical models for aircraft trajectory design: A survey’. In: *Air Traffic Management and Systems*. 2014, pp. 205–247 (cited on page 262).
- [325] Sándor P. Fekete, Björn Hendriks, Christopher Tessars, Axel Wegener, Horst Hellbrück, Stefan Fischer, and Sebastian Ebers. ‘Methods for Improving the Flow of Traffic’. In: *Organic Computing - A Paradigm Shift for Complex Systems*. Ed. by Christian Müller-Schloer, Hartmut Schmeck, and Theo Ungerer. Springer, 2011, pp. 447–460. doi: [10.1007/978-3-0348-0130-0_29](https://doi.org/10.1007/978-3-0348-0130-0_29) (cited on page 262).
- [326] Michael Schreckenberg and Reinhard Selten (editors). *Human Behaviour and Traffic Networks*. Springer, 2004 (cited on page 262).
- [327] Jacob T. Schwartz and M. Sharir. ‘On the piano movers’ problem: III. Coordinating the motion of several independent bodies: the special case of circular bodies moving amidst polygonal barriers’. In: *Int. J. Robotics Res.* 2.3 (1983), pp. 46–75 (cited on page 262).
- [328] J. E. Hopcroft, J. T. Schwartz, and M. Sharir. ‘On the Complexity of Motion Planning for Multiple Independent Objects; PSPACE-Hardness of the Warehouseman’s Problem’. In: *International Journal of Robotics Research* 3.4 (1984), pp. 76–88 (cited on page 262).
- [329] John E. Hopcroft and Gordon T. Wilfong. ‘Reducing Multiple Object Motion Planning to Graph Searching’. In: *SIAM J. Comput.* 15.3 (1986), pp. 768–785. doi: [10.1137/0215055](https://doi.org/10.1137/0215055) (cited on page 262).
- [330] Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Henk Meijer, and Christian Scheffer. ‘Coordinated Motion Planning: Reconfiguring a Swarm of Labeled Robots with Bounded Stretch’. In: *SIAM J. Comput.* 48.6 (2019), pp. 1727–1762. doi: [10.1137/18M1194341](https://doi.org/10.1137/18M1194341) (cited on pages 262–264).

- [331] Kiril Solovey and Dan Halperin. 'k-color multi-robot motion planning'. In: *Int. J. Robotics Res.* 33.1 (2014), pp. 82–97. doi: [10.1177/0278364913506268](https://doi.org/10.1177/0278364913506268) (cited on page 263).
- [332] Stephen Kloder and Seth Hutchinson. 'Path planning for permutation-invariant multirobot formations'. In: *IEEE Trans. Robotics* 22.4 (2006), pp. 650–665. doi: [10.1109/TR0.2006.878952](https://doi.org/10.1109/TR0.2006.878952) (cited on page 263).
- [333] Matthew Turpin, Nathan Michael, and Vijay Kumar. 'Trajectory Planning and Assignment in Multirobot Systems'. In: *Algorithmic Foundations of Robotics X - Proceedings of the Tenth Workshop on the Algorithmic Foundations of Robotics, WAFR 2012, MIT, Cambridge, Massachusetts, USA, June 13-15 2012*. Ed. by Emilio Frazzoli, Tomás Lozano-Pérez, Nicholas Roy, and Daniela Rus. Vol. 86. Springer Tracts in Advanced Robotics. Springer, 2012, pp. 175–190. doi: [10.1007/978-3-642-36279-8_11](https://doi.org/10.1007/978-3-642-36279-8_11) (cited on page 263).
- [334] Aviv Adler, Mark de Berg, Dan Halperin, and Kiril Solovey. 'Efficient Multi-Robot Motion Planning for Unlabeled Discs in Simple Polygons'. In: *IEEE Trans Autom. Sci. Eng.* 12.4 (2015), pp. 1309–1317. doi: [10.1109/TASE.2015.2470096](https://doi.org/10.1109/TASE.2015.2470096) (cited on page 263).
- [335] Kiril Solovey, Jingjin Yu, Or Zamir, and Dan Halperin. 'Motion Planning for Unlabeled Discs with Optimality Guarantees'. In: *Robotics: Science and Systems XI, Sapienza University of Rome, Rome, Italy, July 13-17, 2015*. Ed. by Lydia E. Kavraki, David Hsu, and Jonas Buchli. 2015. doi: [10.15607/RSS.2015.XI.011](https://doi.org/10.15607/RSS.2015.XI.011) (cited on page 263).
- [336] Kiril Solovey and Dan Halperin. 'On the hardness of unlabeled multi-robot motion planning'. In: *Int. J. Robotics Res.* 35.14 (2016), pp. 1750–1759. doi: [10.1177/0278364916672311](https://doi.org/10.1177/0278364916672311) (cited on page 263).
- [337] André Naz, Benoît Piranda, Julien Bourgeois, and Seth Copen Goldstein. 'A distributed self-reconfiguration algorithm for cylindrical lattice-based modular robots'. In: *15th IEEE International Symposium on Network Computing and Applications, NCA 2016, Cambridge, Boston, MA, USA, October 31 - November 2, 2016*. Ed. by Alessandro Pellegrini, Aris Gkoulalas-Divanis, Pierangelo di Sanzo, and Dimiter R. Avresky. IEEE Computer Society, 2016, pp. 254–263. doi: [10.1109/NCA.2016.7778628](https://doi.org/10.1109/NCA.2016.7778628) (cited on page 263).
- [338] Zahra Derakhshandeh, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. 'Universal Shape Formation for Programmable Matter'. In: *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2016, Asilomar State Beach/Pacific Grove, CA, USA, July 11-13, 2016*. Ed. by Christian Scheideler and Seth Gilbert. ACM, 2016, pp. 289–299. doi: [10.1145/2935764.2935784](https://doi.org/10.1145/2935764.2935784) (cited on page 263).
- [339] Seth Copen Goldstein and Todd Mowry. 'Claytronics: A scalable basis for future robots'. In: *Robosphere*, Nov (2004) (cited on page 263).
- [340] Pierre Thalamy, Benoît Piranda, and Julien Bourgeois. 'Distributed Self-Reconfiguration using a Deterministic Autonomous Scaffolding Structure'. In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019*. Ed. by Edith Elkind, Manuela Veloso, Noa Agmon, and Matthew E. Taylor. International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 140–148 (cited on page 263).
- [341] Loïc Crombez, Guilherme Dias da Fonseca, Yan Gerard, Aldo Gonzalez-Lorenzo, Pascal Lafourcade, and Luc Libralesso. 'Shadoks Approach to Low-Makespan Coordinated Motion Planning (CG Challenge)'. In: *37th International Symposium on Computational Geometry, SoCG 2021, June 7-11, 2021, Buffalo, NY, USA (Virtual Conference)*. Ed. by Kevin Buchin and Éric Colin de Verdière. Vol. 189. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 63:1–63:9. doi: [10.4230/LIPIcs.SoCG.2021.63](https://doi.org/10.4230/LIPIcs.SoCG.2021.63) (cited on page 267).
- [342] Hyeyun Yang and Antoine Vigneron. 'A Simulated Annealing Approach to Coordinated Motion Planning (CG Challenge)'. In: *37th International Symposium on Computational Geometry, SoCG 2021, June 7-11, 2021, Buffalo, NY, USA (Virtual Conference)*. Ed. by Kevin Buchin and Éric Colin de Verdière. Vol. 189. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 65:1–65:9. doi: [10.4230/LIPIcs.SoCG.2021.65](https://doi.org/10.4230/LIPIcs.SoCG.2021.65) (cited on page 267).

- [343] Paul Liu, Jack Spalding-Jamieson, Brandon Zhang, and Da Wei Zheng. ‘Coordinated Motion Planning Through Randomized k-Opt (CG Challenge)’. In: *37th International Symposium on Computational Geometry, SoCG 2021, June 7-11, 2021, Buffalo, NY, USA (Virtual Conference)*. Ed. by Kevin Buchin and Éric Colin de Verdière. Vol. 189. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 64:1–64:8. doi: [10.4230/LIPIcs.SocG.2021.64](https://doi.org/10.4230/LIPIcs.SocG.2021.64) (cited on page 267).