Master's Thesis

Algorithmic methods for complex dynamic sweeping problems

Dominik Krupke

November 21, 2016

Institute of Operating Systems and Computer Networks
Prof. Dr. Sándor Fekete

Supervisor:
Prof. Dr. Sándor Fekete

Statement of Originality

This thesis has been performed independently with the support of my supervisor/s. To the best of the author's knowledge, this thesis contains no material previously published or written by another person except where due reference is made in the text.

Braunschweig, November 21, 2016

_____

Abstract

Most research on touring and covering problems focuses only on distance costs, but in practice many of these problems have significant turn costs. Adding turn costs to a problem can make it considerably harder, e.g. when considering only distance costs, the minimum 2-factor can be obtained in polynomial time, but it becomes NP-hard with turn costs [1]. We investigate the problem variants of finding a minimum cost cycle cover resp. tour for full coverage (every point has to be covered), subset coverage (specific points have to be covered), and penalty coverage (no point has to be covered but every uncovered point has an individual penalty) in grid graphs as well as in polygonal environments. We first show that finding a minimum turn cycle cover in 2-dimensional grid graphs is already NP-hard, solving the open *Problem* 53 in *The Open Problems Project* [2]. This also implies the hardness of most other considered problems in this thesis. On the positive side, we propose constant factor approximation algorithms for all considered problem variants in grid graphs. For full coverage in more general grid graphs (e.g., hexagonal grids) our approximation technique results in a better approximation factor than the one of Arkin et al. [3]. To the best of our knowledge, we provide the first approximation algorithms for the other problem variants. Our approximation technique for grid graphs can be extended to polygonal environments where every point is only allowed to be covered in a specific amount of orientations. Aggarwal et al. [1] provide an $O(\log n)$-factor approximation algorithm for the non-discretized full coverage problem without obstacles. Our approximation algorithms, for topographies with obstacles, provide approximation factors which do not depend on the problem size, but only on the resolution of the discretization for the full coverage as well as penalty coverage variant. We also provide different integer programs to obtain optimal solutions. In our experiments we were able to increase the optimal solvable problem size to around 700 points for full coverage in 2-dimensional grid graphs, compared to the previous results of around 70 points by de Assis and de Souza [4]. For use with a drone equipped with an electrical lattice to hunt mosquitoes, we further propose two simple practical heuristics for planing tours which try to maximize the expected catch ratio, based on the success of previous runs with limited energy.

Preface

This thesis has a webpage `http://krupke.cc/m/tubs/mt/` to provide updates, an errata, and additional material. The errata hopefully remains empty as the results are all carefully checked but in case you want to use some of the results, please check it first. It is likely that there are at least some misleading/unclear sentences or some forgotten details due to the time pressure. During the proofreading, I also encountered some errors that probably have been created by unknowingly pressing some key in Vim. All the content of the webpage can also be found on `https://github.com/d-krupke/mt` in case the first link becomes invalid (which unfortunately happens too often to links in scientific publications).

A master's thesis are six months of endless work that conclude five years of studying and there are a few people to thank for their support.

- First of all, Maximilian Ernestus for your mental support, feedback, and proof reading despite being at the other end of Germany.
- Sebastian Morr, has been a nice time sharing an office with you.
- Marco Nikander, for mental support and proof reading. Also some nice evenings in the Schuntille with the other guys during the last years.
- Aaron Becker, for your help with the problem motivation (and especially motivating me).
- Sándor Fekete for supervising me for such a long time (including this thesis) and helping me to collect experiences (e.g., Dagstuhl, EuroCG,...). Also the rest of the IBR Alg, namely (alphabetically) Victor Alvarez, Anna Jankowski, Phillip Keldenich, Ute Marchot, Frank Quedenfeld, Christian Rieck, Christian Scheffer, and Arne Schmidt, for providing such a good environment. I am looking forward to the next years with you.
- Michael Hemmer, who unfortunately left the institute a year ago but still had a lot of influence on me (e.g., EuroCG, IROS, SEA, and your great lectures on computational geometry).

The points above have no real order and I probably also forgot someone.

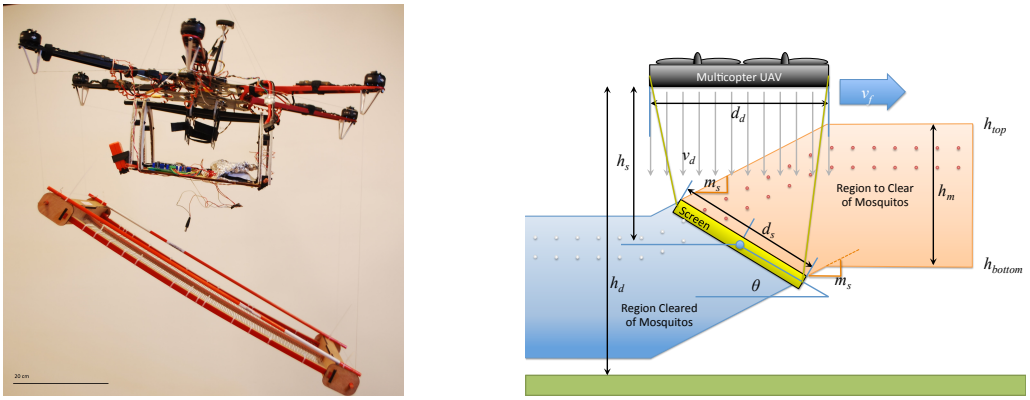
Contents

List of Figures

List of Tables

1 Introduction

Finding tours of minimum cost is one of the most important and well-researched problems in combinatorial optimization. It has many practical applications as minimizing the time needed to deliver parcels or to switch between a set of configurations in a production process. Also, geometric covering problems like mowing the grass can be considered as a combinatorial tour problem by partitioning the area into fields of the size of the mower and searching for a tour that visits all these fields. Most of the research has been focused on minimizing the length of the tour, i.e., the sum of edge weights in the underlying graph. This is also known as the *traveling salesman problem (TSP)*. However, many problems include significant additional costs, such as turn costs, for switching between two edges. For example, you can mow a straight line much faster and with less effort than some zig-zag path of the same length.



(a) (b)

Figure 1.1: A drone equipped with an electrical lattice to hunt mosquitoes. Images by Aaron Becker and his team. Reprinted with permission.

Our main motivation in this thesis is to find good tours for a drone equipped with an electrical lattice to hunt mosquitoes. A picture of such a drone that is currently developed by Aaron Becker and his team at the University of Houston, Texas, can be seen in Fig. 1.1. By nature, a drone has to slow in order to make a controlled turn. The payload of the lattice hanging below the drone additionally increases this problem. Further, these drones can not fly infinitely but have a limited battery such that the objective for large areas becomes to catch as many mosquitoes as possible instead of touring with minimum cost over the whole area. Of course, there are also many other motivations, for instance, planning tours for cleaning robots. Nonetheless, only Chapter 6 is practice-oriented while the primary results of this thesis in Chapter 4 and Chapter 5 are mainly of theoretical nature.

We first consider tours and cycle covers in grid graphs in Chapter 4 which is the primary chapter of this thesis. Let us assume we want to find a coverage tour for an integral orthogonal polygon and can cover a unit square at once. This area can also be seen as a set of fields for which we can build an adjacency graph (the grid graph) by putting a square grid onto it. This transformation has been visualized in Fig. 1.2. A tour that visits

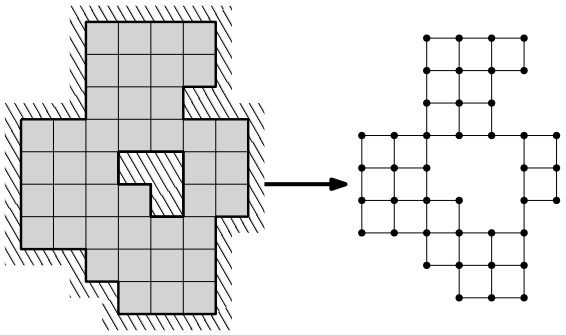


Figure 1.2: Converting the coverage area into a grid graph

all vertices in this grid graph is automatically a coverage tour for the polygon. One can actually show, that a shortest tour in the grid graph is also a shortest coverage tour in the polygon if the movement is restricted to be orthogonal, as shown by Arkin et al. [5]. Of course in practice, the area to be covered is seldom a perfect integral orthogonal polygon, but one can imagine that this is often still a sufficiently good approximation for a mower or a cleaning robot. Actually, the important part for these scenarios are anyway the large free areas for which the grid works fine and the boundary that possibly did not fit into the grid can be covered by an additional boundary sweep. If we have turn costs, following a diagonal boundary is, of course, more critical than with only distance costs. In a square grid, we would have to make a 90° turn for every field. Hence, using a hexagonal grid instead of a square grid might be a better approximation in some scenarios. A hexagonal grid leads to a graph where every vertex has six neighbors with an angle difference of 60° as can be seen in Fig. 1.3. Also, 3-dimensional grid graphs are possible and considered in

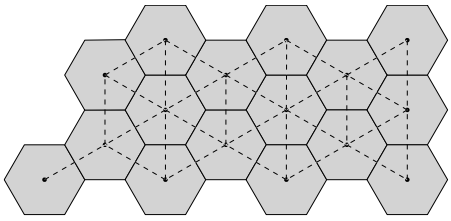


Figure 1.3: Hexagonal fields lead to a hexagonal grid graph (dashed) where each vertex can have up to six neighbors

this thesis.

The most basic problem is to find a tour with minimum costs that visits all vertices (full coverage). With only distance costs, this is also known as the traveling salesman problem

in grid graphs and has been shown to be NP-hard [6]. The cycle cover problem is a relaxation of this problem and only searches for a set of cycles such that every field is covered by at least one cycle. With only distance costs, this problem is solvable in polynomial time via a 2-factor. In a 2-dimensional grid graph (square grid) there can be straight passings, 90° turns (simple turns), and 180° turns (u-turns). In this thesis, we try to find tours and cycle covers with minimum costs where the cost is a weighted sum of the length/distance and the number of turns (u-turns counting as two simple turns).

Furthermore, we not only consider *full coverage tours and cycle covers* but also two other variants: *subset coverage* and *penalty coverage*. There might only be some fields that need coverage and some further fields that are only possible connections. For the mower example, these fields might be the grass while the other fields are a trail. For the cleaning robot, we might have known dirty areas and clean areas. For the mosquito hunting drone, there might be areas favored by mosquitoes and areas with low expected mosquito densities. In *subset coverage tours and cycle covers*, we only demand that at least these important fields are covered but the other fields can also be used. For the last two examples, the importance is relative and we would possibly not want to be too strict on enforcing a coverage. In *penalty coverage*, we assign each vertex a penalty that has to be paid if it is not covered. This results in 'biggest bang for the buck' tours and cycle covers where a less important field is only covered if it can be cheaply integrated and even an important field might not be covered if the additional costs for integrating it are too high. A formal description of all the considered problems in grid graphs is given in Sec. 4.1.

Arkin et al. [3] have already shown that finding minimum turn tours in grid graphs is NP-hard even if a minimum turn cycle cover is given. The NP-hardness follows trivially for full coverage tours, subset coverage tours, and penalty coverage tours with weighted distance and turn costs. The question whether the minimum turn cycle cover can be calculated in polynomial time like the minimum length cycle cover or is already NP-hard remained open until now and is listed as *Problem* 53 in *The Open Problems Project* edited by Demaine, Mitchell, and O'Rourke [2]. In the meantime, Maurer [7] has shown that the *minimum turn cycle partition*, where every vertex has to be covered exactly once, can be calculated in polynomial time and stated the assumption that *minimum turn cycle cover* is NP-hard. In Sec. 4.2 we show that minimum turn cycle cover in planar grid graphs is indeed NP-hard, answering the above mentioned open problem. The NP-hardness of the other two cycle cover variants in grid graphs follows trivially.

Because all considered problem variants are NP-hard, there cannot exist optimal polynomial time algorithms for any of them (assuming $P \neq NP$). Instead, we provide constant factor approximation algorithms for all problem variants, i.e., algorithms that guarantee to provide solutions in polynomial time that cost at most a specific factor more than the optimal solution, in Sec. 4.3. While we are neither able to improve the approximation factor nor the running time of Arkin et al. [3] for full coverage tours and cycle covers in 2-dimensional grid graphs (square), our method has a better guaranteed approximation factor for more general grids such as hexagonal or 3-dimensional grid graphs. To the best

of our knowledge, we provide the first approximation algorithms for the subset coverage and penalty coverage variants.

Obtaining optimal solutions for all problem variants using integer programming is considered in Sec. 4.4. Modern integer programming solver as CPLEX [8] have shown to be powerful tools for solving many hard problems to optimality. Because the performances of different formulations for different instance types can vary, we consider and compare multiple such formulations. We are able to solve 50% of our experiment instances of full coverage tours and cycle covers with only turn costs of size 700 resp. 1000-1500 within 15 min on a modern desktop computer. The largest optimally solved instance had 2900 fields. This increases the solvable instance size by a significant factor since de Assis and de Souza [4] were only able to solve instances with a size of around 70 vertices. Interestingly, calculating an optimal cycle cover seems to be the hardest part in calculating an optimal tour. The formulations of the full coverage variant can be easily adapted to subset coverage and penalty coverage. Furthermore, we were able to solve 50% of the instances of size 900 resp. 700 for subset coverage cycle cover resp. tour with 25% subset fields and instances of size 1400 resp. 800 for penalty coverage cycle cover resp. tour. The experimental evaluations can be found in Sec. 4.5.

The limitation of the movement to the grid might be too restrictive for some scenarios. Hence, we also consider a more geometric problem variant in Chapter 5. In it, we have a set of points in $\mathbb{R}^2$ given as well as some polygonal obstacles. Additionally, for every point in this set we have a set of orientations given in which this point can be passed. Wanted is a polygonal tour of minimum costs that passes all fields in one of its orientations and does not intersect the obstacles. The cost consists of a weighted sum of the length and turning angles. An example can be seen in Fig. 1.4.

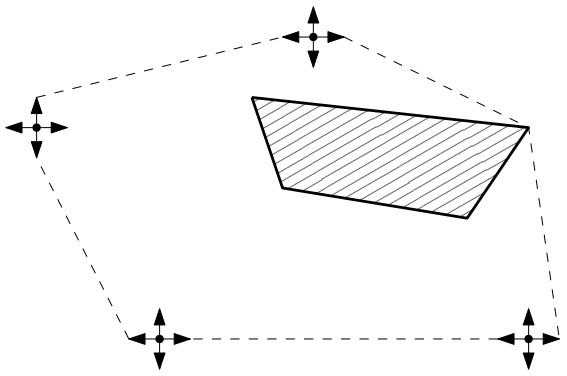


Figure 1.4: Every point has a set of orientations it can be passed through. The connection between two subsequent points can be any polygonal path that does not intersect the obstacles and the cost of a tour consists of the length of this connection and the sum of turn angles. A possible tour is drawn by a dashed path.

For this problem variant, we also consider penalty coverage but not subset coverage since subset coverage does not make any sense here. The formal definition of all considered problems in this geometric setting is given in Sec. 5.1. Our approximation method

for grid graphs can easily be generalized to approximation algorithms for full coverage and penalty coverage cycle covers and tours where the number of orientations per point is limited, as shown in Sec. 5.3. The approximation factor only depends on the resolution of this passing orientation discretization. If the passing orientations are not discretized, only an $O(\log n)$ approximation algorithm by Aggarwal et al. [1] is known that does not include obstacles. Our approximation algorithm needs to calculate the minimum cost path between two configurations as a subroutine. We show in Sec. 5.2 that this is possible in polynomial time by using the visibility graph similar to the euclidean shortest path. Stating an integer program to obtain optimal solutions is straightforward and done in Sec. 5.4. Finally, we provide some rough experiments for the full coverage integer program and approximation algorithm in Sec. 5.5. The discretization increases the optimal solvable instance size compared to a size of fewer than 100 points for the continuous version by Aichholzer et al. [9] but interestingly it is already only slightly better for a discretization to orthogonal and diagonal passing, i.e., 4 orientations. For only orthogonal passings (i.e., 2 orientations), we are able to solve instances with 300 points to optimality. In our experiments, the approximation algorithm returns solutions that are relatively close to the optimum with the worst solution being 1.484 times the optimum; hence, far better than the proven ratio.

A practical approach for the mosquito hunting drone is proposed in Chapter 6. It is not only much faster than the above-mentioned results but also includes a maximum runtime of the drone. Further, this scenario is difficult to model theoretically and there is no use to find the best tour in such a model if it possibly only exploits some shortcomings of it. Even if we would have all information, a nearly accurate model of the mosquitoes would probably also be too expensive for serious optimization. In our approach, we put a density map in form of a square grid that represents the expected mosquito densities in every field onto the environment. As the drone can measure when it catches mosquitoes, this map can be updated after each run, and thus learn to some degree. On the first run, the density map can have a uniform density and then get more accurate after each run. The environment is approximated by polygons (e.g. a tree by a bounding hexagon) and we assume it to be relatively simple. We consider two heuristics which are similar to the common TSP heuristics *Cheapest Insertion* and *Nearest Neighbor*. The main idea is to sample points based on the densities and extend the tour by the best sample. The two heuristics have been implemented to obtain a first impression of their performance but a serious evaluation and a more advanced implementation with additional optimizations are left to future work.

A short introduction into the used methods, common definitions and notations, and corresponding literature references are given in Chapter 2. Less common definitions and notations are provided where they are needed to ease reading of single sections. A summary of previous and related work is given in Chapter 3. Chapter 7 concludes this thesis with a summary and future work.

2 Preliminaries

This chapter provides a quick repetition of the general definitions and notations including references to corresponding textbooks. It is meant as an anchor for readers interested in the problem but unfamiliar with some subjects. It of course also helps to assess the point of view that underlies this thesis. The experienced reader that is familiar with the basics of graph theory, geometry, computational complexity, and integer programming may skip the chapter and only return if questions arise in the main chapters. All uncommon definitions are provided when and where they are needed in the main chapters.

2.1 Approximation algorithms and NP-hardness

We differ between three types of algorithms/methods regarding their solution quality in this thesis: *optimal algorithms*, *approximation algorithms*, and *heuristics*. Optimal algorithms always return, as the name implies, optimal solutions or detect that no such solution can exist. The corresponding runtime for hard problems might be exponential or worse regarding the input size. For these hard problems, (polynomial time) approximation algorithms are desired. Approximation algorithms guarantee that their solution is always within a specific factor of the optimum. This factor is called the *approximation factor*. A 2-approximation algorithm for a minimization problem would guarantee that the returned solution has at most twice the objective value of the optimum. The approximation factor can also depend on parameters of the algorithm or properties of the input instance, e.g. its size. If the approximation factor is constant, we call the corresponding algorithm a constant factor approximation algorithm. In this thesis, we always refer by approximation algorithm to polynomial time algorithms even if this is not explicitly mentioned. An approximation factor is an upper bound (minimization problem) on the worst case. Often the average solutions are much better and even the worst case sometimes can be far below the proven factor. A heuristic, in contrast, refers to a procedure that may or may not return a solution with no guarantee on its quality (sometimes not even on its runtime). A useful heuristic, however, is designed to work well in practice even if no theoretical guarantees can be given. Obviously, every optimal algorithm is also a (possibly non-polynomial time) approximation algorithm and an approximation algorithm is also a heuristic. An excellent textbook on approximation algorithms has been written by Williamson and Shmoys[10].

In this thesis, we refer by a *hard* problem to a problem in the NP-hard class. A problem is NP-hard if it can be reduced to another NP-hard problem (e.g., *Satisfiability* or *Set Cover*) in polynomial time. This means that we can transform any instance of some NP-hard problem in polynomial time to an instance of our problem and its corresponding optimal solution can be transformed in polynomial time to an optimal solution of the origin problem. This means that if we have a polynomial time algorithm for our problem, we

also have a polynomial time algorithm for the problem we reduced our problem to. A problem is NP-complete if it is NP-hard and a solution can be verified in polynomial time (e.g., *Satisfiability* is NP-complete because it is NP-hard and we can easily check if a variable assignment evaluates a boolean formula to true). Problems for which a polynomial time optimal algorithm exists are in the class P. It is an open problem if $P = NP$ but the common opinion is that $P \neq NP$, i.e., there cannot exist a polynomial time optimal algorithm for any problem in the NP-hard class. More details can be found in the textbook of Hopcroft et al. [11].

An introduction to the common asymptotic O-notation, that is used for complexity analysis, can be found in the book of Cormen et al. [12].

2.2 Graph theory

This thesis contains a lot of graph theory and the reader should have some basic knowledge, such as can be found in the book of Diestel [13]. As there are different common notations, we now quickly repeat the notations used in this thesis.

An (undirected) graph $G(V, E)$ consists of a set of vertices V and a set of edges E where an edge $e = \{u, v\}$ connects the vertices $u \in V$ and $v \in V$. A weighted graph $G(V, E)$ additionally has a function $c : E \to \mathbb{R}$ that assigns a weight to each edge. A path in a graph $G(V, E)$ between two vertices u and v is a sequence of vertices $w_0 = u, w_1, \ldots, w_m = v$ with $\forall i = 0, \ldots, m-1 : \{w_i, w_{i+1}\} \in E$. The cost of this path is $\sum_{i=0,\ldots m-1} c(\{w_i, w_{i+1}\})$. In case a graph is unweighted, a default weight of 1 is assigned to each edge. The shortest path or minimum cost path between u and w is a path between u and v such that no path between u and w with lower costs exist. A graph is connected if there exists a path between any two vertices. A directed graph $G(V, E)$ is defined analogously but the edges (also called arcs) are directed ($e = (u, v) \in E$ allows a transition from u to v but not from v to u). For example, an u-v-path becomes a sequence $w_0 = u, w_1, \ldots, w_m = v$ with $\forall i = 0, \ldots, m-1 : (w_i, w_{i+1}) \in E$.

The set of neighbored vertices to a vertex v is denoted by $N(v)$. The set of edges between two vertex sets U and W are denoted by $E(U, W)$ and $E(v)$ returns all edges incident to v. We also use notations as $V'(x)$ to describe all vertices in the vertex set V' that are correlated to some x (apparent from the context).

The traveling salesman problem (TSP) searches for a tour that visits every vertex in a graph exactly once while the edge weights are minimal. The 2-factor (or 2-matching) on the other hand searches for a set of tours with minimum costs such that every vertex is visited exactly once. A minimum weight perfect matching searches for a set of edges such that the edge weights are minimal and every vertex is in exactly one edge.

A (2-dimensional) grid graph is a graph where every vertex is an integral 2-dimensional point and there is an edge between two points if and only if their distance is one. A 3-dimensional grid graph can be defined analogously. There are also other grid graphs as hexagonal grid graphs which we do not describe here but can be seen, e.g., in Fig. 1.3. If we speak of a grid graph, we usually refer to a 2-dimensional grid graph. A strip in a grid

graph is a straight path, a full strip is a strip that is not fully contained in any other strip.

2.3 Geometry

Especially in Chapter 5, we have to deal with elements of computational geometry. A nice textbook on this subject has been written by De Berg et al. [14].

We only consider simple 2-dimensional environments with polygonal objects, i.e., objects that consist of segments and points. A segment is a line between two points. A simple polygon is a closed chain of segments which do not self-intersect, a (non-simple) polygon might also have some simple polygons (holes) within its area. A polygon is convex if it does not have any concave points. In this thesis, we assume the area of a polygon to be open, i.e., the boundary does not belong to the interior. A polygonal path is a chain of incident segments between two points and can also be considered as a sequence of points.

The turning angle between two incident segments is the minimum angle between them subtracted from 180°, see Fig. 2.1.

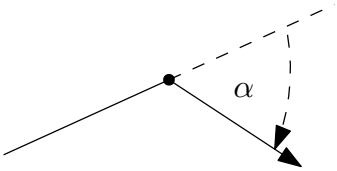


Figure 2.1: Turning angle between two incident segments

2.4 Linear/integer programming

Linear/integer programming (LP/IP) is one of the most important tools in combinatorial optimization for theory as well as practice. There are many introductory textbooks such as the book by Schrijver [15] or Papadimitriou and Steiglitz [16]. Many combinatorial problems can easily be expressed as integer programs and often be solved to optimality by IP-solvers. A linear program with n variables $x = (x_1, \ldots, x_n)^T \in \mathbb{R}^n$ consists of a linear optimization function $f(x) = c^T x$ (with $c \in \mathbb{R}^n$) that has to be minimized or maximized and a set of m linear constraints of the form $a_i^T x \sigma b_i, i \in 1, \ldots, m$ with $\sigma \in \{\leq, \geq, -\}$ and $a_i \in \mathbb{R}^n$. An integer program is a linear program but the variables are restricted to be integral. If we mix these two variable types, we have a mixed integer program but we will also consider them as integer programs, and thus make the integer property of integer programs optional. While linear programs can be solved in linear time, integer programming can easily be shown to be NP-hard since many NP-hard problems can be expressed as integer programs.

We give an example on the traveling salesman problem which can be stated for a graph

$G(V, E)$ with edge cost c_e for edge $e \in E$ as follows:

$$\begin{aligned} \min \quad & \textstyle\sum_{e \in E} c_e * e && && (2.1)\\ \text{s.t.} \quad & \textstyle\sum_{e \in E(v)} e = 2 && \forall v \in V && (2.2)\\ & \textstyle\sum_{e \in E(V', V \setminus V')} e \geq 2 && \forall V' \subsetneq V, V' \neq \emptyset && (2.3)\\ & e \in \{0, 1\} && && (2.4) \end{aligned}$$

An edge is considered as a variable and can have the value 0 if it is not used or 1 if it is used. Hence, for all used edges the cost is added to the sum while for all unused edges the product becomes zero. Every vertex has to have two used edges (Eq. 2.2) and every real subset that is not empty has to have two edges that leave it (Eq. 2.3). Note, that there is an exponential amount of constraints in Eq. 2.3 and most of them are not needed. A common approach is to not use all of these constraints but instead iteratively solve the problem and only selectively add such a constraint to the formulation for every subtour in the intermediate solutions until we obtain a tour. The initial version of this iteratively built integer program equals the 2-factor and can be solved optimally in polynomial time.

In an LP-relaxation, we drop the integrality constraints of the integer program. The problem now becomes efficiently solvable, but, e.g., for the TSP we can obtain partial edges that are, e.g., selected to 40%. For some cases like the matching problem, the solution of the LP-relaxation can be shown to contain only values in $\{0, 0.5, 1\}$, and hence is called half-integral.

Linear programming can also be used to obtain polynomial time approximation algorithms. For example for a set cover variant where every element is in at most k sets, we can easily produce a k-approximation algorithm. Let S be the set of sets and I be the set of items to be covered by a subset of S. The corresponding integer program can be relaxed to the following linear program whose costs are at most the cost of the integer program.

$$\begin{aligned} \min \quad & \textstyle\sum_{s \in S} s && && (2.5)\\ \text{s.t.} \quad & \textstyle\sum_{i \in s, s \in S} s \geq 1 && \forall i \in I && (2.6)\\ & 0 \leq s && \forall s \in S && (2.7) \end{aligned}$$

As we know that there are only at most k elements in the sum on the left side of the constraint and the value of the sum is ≥ 1, we know that at least one set is chosen with at least $1/k$. If we multiply the solution by k, we obtain a fractional solution with k times the cost and, since the LP-relaxation is a lower bound on the optimum, at most k times the optimal cost. In this solution, every element is covered by at least one set that is selected with ≥ 1. Rounding down the solution provides a feasible solution for the problem that is at most k-times more expensive than the optimal solution. Making the fractional solution integral is usually not as simple as in this case. The approximation algorithms in this thesis use a similar technique.

3 Previous and related work

This chapter contains a collection of references to previous and related work. It is not mandatory for an understanding of the later content in this thesis but helps to assess the current state of art and to find further material.

3.1 Traveling salesman problem

The most common tour optimization problem is the *traveling salesman problem (TSP)*. In it, a weighted graph is given and a tour that visits every vertex exactly once with a minimum sum of edge weights is wanted. The traveling salesman problem is NP-hard and without additional constraints on the edge weights, no approximation algorithm can exist for it [17]. For specific classes of graphs, approximation algorithms can be given but even in simple graphs as grid graphs, the problem remains NP-hard (see [6]). If the edge weights fulfill the triangle inequality, a simple 1.5-approximation by Christofides [18] is known that mainly uses a minimum spanning tree and a minimum weight perfect matching to make the tree eularian. Dantzig et al. [19] provided an integer program and solved an instance with 49 vertices to optimality (in 1954). Today, powerful solver such as Concorde [20] exist that, for example, was able to solve a (non-trivial) instance with 85 900 vertices to optimality. For the euclidean traveling salesman problem where all vertices are points in the euclidean space and the edge weights are the euclidean distances, an arbitrary good approximation factor can be obtained by the *polynomial time approximation schemes (PTAS)* by Mitchell [21] or Arora [22]. However, they are already for relatively bad approximation factors impractical.

A beginner friendly book on the traveling salesman problem is by Cook [23]. A more advanced textbook on solving the TSP is by Applegate et al. [24].

3.2 Covering in grid graphs

Chapter 4 of this thesis considers variants of tours (a field might be visited multiple times) in grid graphs (or polyominos) with weighted turn and length costs. This problem equals a variant of orthogonal milling tours with turn costs.

Arkin et al. introduce in [25, 3] the milling with turn costs problem and give a bunch of techniques and results for it. They first show the hardness of finding an optimal tour even in thin 2-dimensional grid graphs with given optimal cycle cover. If a cycle cover of cost c in such a 2-dimensional grid graph is given, a tour of cost $1.5 * c$ can be derived by connecting the cycles. They give a 2.5-approximation algorithm for obtaining a cycle cover, resulting in a 3.75-approximation algorithm for tours. The complexity of finding an optimal cycle cover in a 2-dimensional grid graph was still open until now and listed as *Problem* 53 in *The Open Problems Project* [2]. In this thesis, we solve the problem by showing

the NP-hardness. They also show that a field has only to be visited at most 4 times and the length can otherwise be easily reduced without increasing the number of turns. They give an approximation algorithm for a more general version where the underlying graph does not have to be a grid graph. This algorithm has an approximation ratio of $2\delta + p$ for cycle cover and $2\delta + p + 2$ for tours where δ is the maximum degree of the graph and p is the maximum number of directions. We are able to beat this results in many cases. They also give a PTAS whose running time exponentially depends on the number of holes and the weighting of the turns. Note that their work pays attention to the running time regarding the real instance size which can be much smaller than the number of fields. This is neglected in this thesis.

Maurer [7] proves in his diploma thesis (German) that a cycle partition with a minimum number of turns in grid graphs can be calculated in polynomial time. A point is only allowed to be covered once in a cycle partition, and hence cycle partitions only exist in grid graphs with a 2-factor. He assumes that the cycle cover problem, which is always existent and possibly has fewer turns, is NP-hard. He performed experiments for cycle cover with instances of size less than $50 * 50$ for grid graphs and less than 50 points for general points in $\mathbb{R}^2$. His IP-formulation equals the core of one of our formulations.

For the problem in 2-dimensional grid graphs, de Assis and de Souza [4] provide an optimal IP-formulation and a heuristic. They performed experiments with instances of size less than 76 vertices for the exact IP-formulation. For the heuristic, instances with less than 500 vertices were tested. Note that the exact IP-formulation proposed in this thesis can solve many instances with more than 700 vertices in reasonable time.

For the abstract version on graphs, Fellows et al. [26] researched the parameterized complexity and show that the problem is fixed-parameter tractable by the number of turns, tree-width, and maximum degree.

In her dissertation, Benbernou [27] considered milling with turn costs on the surface of polyhedrons in the 3-dimensional grid. She gives a corresponding 8/3 approximation algorithm for tours.

Related to the milling problem, where the covering tour is not allowed to leave the polygon, is the mowing problem where the polygon is allowed to be left. When this problem is discretized to a set of points to be visited, it is called the *minimum bends traveling salesman problem*. Stein and Wagner [28] give a 2-approximation algorithm on the number of turns if still only orthogonal movement is allowed.

If only the traveled distance is considered, Arkin et al. [5] provide approximation algorithms for the milling and mowing problem. 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 [29] 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 which could also be replaced by, e.g., the common al-

gorithm of Christofides [18]. This problem has also been considered by Iwano et al. [30] but with the name 'the traveling cameraman problem'.

3.3 Milling/covering in practice

An extensive survey on (mostly heuristic) covering paths/tours methods for robots has been made by Galceran and Carreras [31]. A possible technique is to partition the area by a grid as, e.g., done by Zelinsky et al. [32], or Gabriely and Rimon [33]. The grid graphs considered in this thesis are assumed to origin from such a discretization. A different approach is to partition the area into simpler geometric areas (e.g. trapezoidal map) and then use simple patterns to cover the simple areas. For example, Huang [34] does such a partition and then calculates the optimal orientation for a zig-zag coverage of these simple areas. This correlates to the number of turns needed for the zig-zag coverage.

Covering tour optimization is also important for cutter. Yao et al. [35] analyze different cutter path generation methods and propose a greedy and a branch and bound algorithm for generating so-called hybrid cutter paths that use different path types for different parts. They also mention the increase of the costs due to sharp turns.

Covering path planning is also important on larger scales, such as for autonomous agricultural machines on fields, whose high precision GPS allow them to follow previously optimized tours. Corresponding work has been done by, e.g., Taïx et al. [36] and Ali et al. [37]. Ali et al. [37] consider tour planning for tractors for crop harvesting operations including turn penalties.

Covering is also done by UAVs for area observation. Some UAVs have strong restrictions on the turn radius that usually cannot be ignored for covering tour planning. Such optimizations have been considered by, e.g., Ahmadzadeh et al. [38] and Agarwal et al. [39].

3.4 Quadratic TSP

Chapter 5 generalizes the problem from grid graphs to points in the $\mathbb{R}^2$ plane with polygonal obstacles and a limited discrete set of orientations a point can be passed through. The cost consists of the weighted sum of turning angles and the length. While in the grid graph only orthogonal movement (90° and 180° turns) were allowed, now all turns (0° - 180° clockwise or counterclockwise) are allowed and weighted linearly. This is related to the *quadratic traveling salesman problem (QTSP)* where the cost of an edge depends on the previous edge (the costs are not induced by geometry but can be arbitrarily selected).

Jäger and Molitor [40] give two exact solution methods and seven heuristics for the asymmetric version. They were only able to give exact solutions for instances with 24 points and heuristic solutions for 44 points. They also noted that random instances seem to be simpler than real instances.

Fischer et al. [41] give similar but more advanced results on the symmetric version. Their branch and cut integer programming approach (with problem specific cutting planes) is able to calculate optimal solutions for real-world instances (from bioinformatics) of the size of up to 100 vertices in less than 10 minutes.

An extensive study of the QTSP polytope has been made by Fischer in her dissertation [42] (see also [43]).

More computational results with integer programming containing also more variants (e.g., the maximization instead of minimization) follow in Aichholzer et al. [9]. The tested instances have sizes of less than 100 points.

Rostami et al. [44] also consider optimal solver and lower bounding procedures for the Q(A)TSP but the experiments only consider very small instances (less than 25 points).

If the instances are in the $\mathbb{R}^2$ plane and only the turning angles are measured, the problem is called the *angular metric traveling salesman problem*. Aggarwal et al. [1] provide an $O(\log n)$ approximation algorithm for cycle covers and tours. In this thesis, we are able to obtain an $O(\omega)$-approximation if the ways to pass a point are discretized to ω orientations (e.g. horizontally and vertically). They also show that optimal angular metric cycle covers and tours are NP-hard to calculate. The NP-hardness proof in Sec.4.2.3 for subset cycle cover in thin 2-dimensional grid graphs is adapted from this proof.

3.5 Angle and curvature constrained tours and paths

Fekete and Woeginger [45] consider the problem of connecting a point-set with a tour where the angles between the two successive edges are constrained. Two successive points are always connected by a straight line, hence turns are only made at the points. They show that it is already NP-hard to check if there exists an axis parallel tour. If only $\pm 90^\circ$ turns are allowed, the problem can be solved in polynomial time.

In many scenarios, there are no sharp turns but the maximum curvature of a path in a geometric environment has to be limited. Finding a curvature constrained shortest path with obstacles has been shown to be NP-hard by Lazard et al. [46]. Without obstacles, the problem is known as the *Dubins path* and all optimal solutions have been characterized by Dubins [47]. They all follow a small amount of simple patterns, and thus can be efficiently calculated. For the case that the path is not allowed to leave a convex polygon, Agarwal et al. [48] provide an efficient algorithm. Agarwal and Wang [49] provide an $1+\epsilon$ approximation algorithm for finding a shortest path in environments with arbitrary polygonal obstacles in $O((n^2/\epsilon^4)\log n)$ time provided the optimal path is robust. In the case that the obstacles are moderate (convex, consisting only of arcs of radius 1 and line segments) the shortest path between two configurations can be found in polynomial time as shown by Boissonnat and Lazard [50]. The running time can be bounded to $O(n^4 \log n)$ or $O(n^2 \log n)$ if the minimum length of line segments in obstacles is bounded. It also works if the obstacles are not disjunct but this increases the running time to $O(n^7)$. Takei et al. [51] consider the solution of the problem from a practical perspective.

The *Dubins traveling salesman problem* is considered by Le Ny et al.[52].

3.6 Related combinatorial problems

We also consider penalty tours in which no field has to be visited but every unvisited field results in a penalty. This variant has also been previously considered for classical combi-

natorial problems. Goemans and Williamson [53] provide a practical approximation technique for constrained forest problems and similar problems. In especially, they provide a 2-approximation algorithm for *prize-collecting Steiner trees* in general symmetric graphs and *penalty traveling salesman problem* in graphs that fulfill the triangle inequality. Their prize-collecting Steiner tree algorithm is used by us, e.g., in Sec. 4.3.4. They call the penalty traveling salesman problem the *prize-collecting traveling salesman problem*. The prize-collecting Steiner tree is actually only the tree version of the penalty traveling salesman problem. An introduction into approximation algorithms for prize-collecting/penalty problems, k-MST/TSP, and minimum latency problems is given by Ausiello et al. [54].

In the problems considered in this thesis, there are different costs for different ways of passing a vertex. This is related to the *generalized traveling salesman problem (GTSP)* where the tour has to visit a vertex of every set. This also allows different ways of visiting a set. By providing a vertex for every way of passing and having asymmetric edges, the quadratic TSP can be modulated. The symmetric GTSP can be transformed into the asymmetric TSP. Helsgaun et al. [55] solved instances of the equality GTSP (every set is visited exactly once) with up to 17 180 clusters and 85 900 vertices close to optimality.

3.7 Related geometric problems

In Chapter. 5 we consider a 2-dimensional polygonal environment where the shortest path (distance and turn costs) between two configurations is a primitive. The used algorithm uses the visibility graph which is also a rational way for the euclidean shortest path problem. The visibility graph can be calculated in $O(n^2)$, e.g., by the algorithm of Asano et al. [56] or Welzl [57]. This is worst case optimal but there exists also an output sensitive algorithm by Ghosh and Mount [58] that runs in $O(n \log n + k)$ where k is the number of edges in the visibility graph. The shortest path can be calculated more efficiently by the algorithm of Hershberger and Suri [59] that is based on wavefront propagation and needs $O(n \log n)$ initialization time. After initialization, it can calculate the shortest path from any point to a fixed point in $O(\log n)$ time. In 3D, the shortest path problem is NP-hard, see Canny [60]. The bicriteria shortest path in the plane the on sum of length and the sum of turning angles is also NP-hard as shown by Arkin et al. [61].

Related to covering tours are watchman routes where a route (of minimum cost) in a polygon has to be found such that every point in the polygon is visible from one point in the tour. For simple polygons, such a route of minimum length can be found in polynomial time [62]. For polygons with holes, the problem is NP-hard [63]. Mitchell [64] gives an $O(\log^2 n)$ approximation algorithm for this problem and a lower bound on the approximateability of $c \log n$ for a constant $c > 0$. Alsuwaiyel and Lee [65] consider watchman tours in simple polygons with minimum number of turns. They show the NP-hardness of the problem and give a 3-approximation algorithm.

4 Grid graph

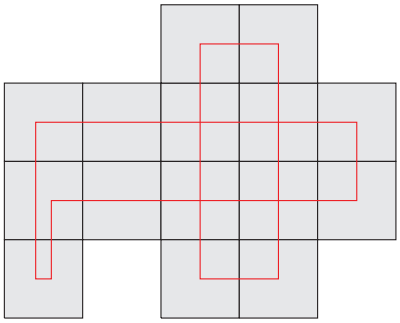
In this chapter, we consider a set of cycle cover and tour problems with turn costs in grid graphs. The focus lies on 2-dimensional grid graphs, but we also consider generalizations to other structures like 3-dimensional grid graphs or hexagonal grid graphs. A cycle cover is the analog of a 2-factor in general graphs, but vertices (fields) might be visited multiple times and a cycle is also allowed to consist of only two fields. Also, the tours are allowed to visit vertices and edges multiple times, contrary to the classical traveling salesman problem. The cycle cover problem is for tours with turn costs of high importance: it contributes a large part to the approximation factors and is the slowest step in integer programming. Until now, it was unknown whether calculating cycle covers with minimum turn costs can be done in polynomial time or is NP-hard and listed as *Problem* 53 in *The Open Problems Project* edited by Demaine, Mitchell, and O'Rourke [2].

We do not only consider the problem where a cycle cover or a tour has to cover all fields but also a variant where only a subset of fields is necessary and a variant where no field has to be visited but every unvisited field contributes an individual penalty. The full coverage version is referenced by the keyword 'full coverage', the other two by 'subset coverage' resp. 'penalty coverage'. The full coverage tour is called 'milling with turn costs' in the literature (e.g. [3]). We often leave out the word 'coverage' and a cycle cover or a tour without any predicate refers to the full coverage variants unless the context implies otherwise. The cost functions in this thesis are a linear combination of turn costs (cost for switching between two edges) and distance costs (cost for using an edge). We use terms as 'penalty cycle cover' instead of more precise names as 'penalty cycle cover in grid graphs with turn and distance costs' because such long names would enforce us to use a lot of similar looking acronyms. To the best of our knowledge, the subset and penalty versions have not been considered before. The formal problem definitions are given in Sec. 4.1 and a set of examples in Fig. 4.1.

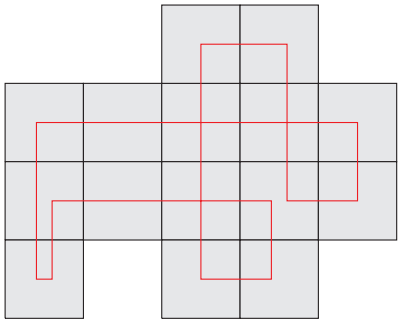
We focus on three elements: computational hardness of the problems, approximation algorithms, and optimal algorithms via integer programming.

Arkin et al.[3] have shown that full coverage tours are NP-hard even in 2-dimensional thin[1] grid graphs in which an optimal full cycle cover can be calculated in polynomial time. In Sec. 4.2.2 we proof that already the full coverage cycle cover problem is NP-hard, answering the open problem in the above mentioned list. As the full cycle cover problem is contained in the subset cycle cover problem and in the penalty cycle cover problem, those problems are NP-hard, too. For thin grid graphs the cycle cover problem was shown to be solvable in polynomial time by Arkin et al. [3]. We show in Sec. 4.2.3 that subset cycle cover is also in thin grid graphs NP-hard.

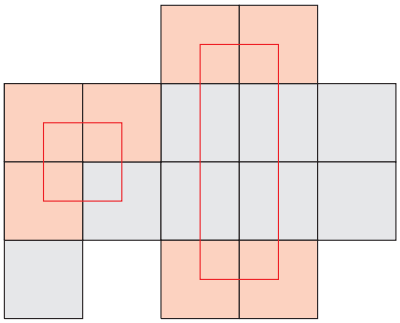
[1] No 2×2 subgraph



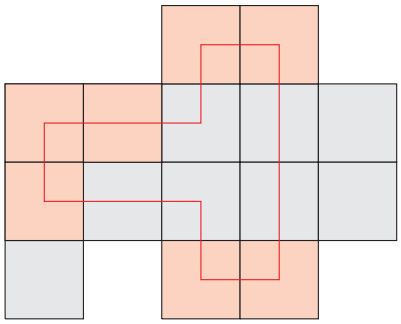
(a) Full cycle cover with 10 turns and length 20



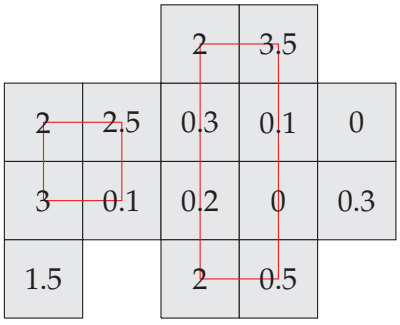
(b) Full tour with 12 turns and length 20



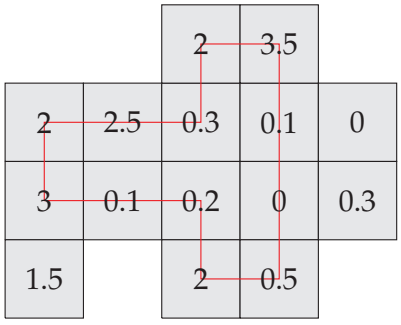
(c) Subset cycle cover with 8 turns and length 12.



(d) Subset tour with 8 turns and length 12.



(e) Penalty cycle cover with 8 turns, length 12, and penalty 1.8



(f) Penalty tour with 8 turns, length 12, and penalty 1.8

Figure 4.1: Examples of the problems. For the subset variant, the necessary fields are highlighted in red. For the penalty variant, the penalties are written into the fields.

	2D	3D	Hexagonal	General grids
Full cycle cover	4^*	6	6	$2 * \omega$
Full tour	6^*	12	12	$4 * \omega$
Subset cycle cover	4	6	6	$2 * \omega$
Subset tour	10	14	14	$4 * \omega + 2$
Penalty cycle cover	6	8	8	$2 * (\omega + 1)$
Penalty tour	16	20	20	$4 * (\omega + 1) + 4$

Table 4.1: Approximation factors of the new approximation techniques. ω is the maximum number of distinct full strips (or orientations) a grid point is covered by. $(*)$ Note that Arkin et al. [3] achieve a factor of 2.5 on the number of turns for cycle cover and a factor of 3.75 on the number of turns for tours in 2-dimensional grid graphs. With distance costs it becomes a 4-approximation for cycle cover and tour.

Because the problems are hard, the next step is to provide polynomial time (on the number of fields/vertices) approximation algorithms that at least achieve a provable factor of the optimal solution. We provide constant factor approximation algorithms for all mentioned problems; an overview can be seen in Table 4.1. The worst cases can be expected to be rather pathologic and the solution quality in practice to be much closer to the optimum than to the proven bounds. Our new technique is not able to beat the strip cover technique of Arkin et al. [3] for 2-dimensional grid graph neither in approximation factor nor in runtime but it is more general and not only works for subset and penalty coverage in grid graphs but also on geometric instances (see Chapter 5). For full coverage in more general grid graphs than 2-dimensional, our new technique also results in a better approximation factor. However, despite the polynomial runtime, the new approximation technique is not practical for large instances due to an internal linear program with cubic tableau size.

Integer programming has become an effective tool to calculate optimal solutions for hard problems. In Sec. 4.4 we provide and evaluate IP formulations for all problem variants. We are able to increase the maximal solvable instance size for full coverage tours by a significant factor compared to the work of de Assis and de Souza [4]. For full coverage cycle cover we can solve similar instance sizes as claimed by Maurer [7] (he only provides some examples but no experimental evaluation). Table 4.2 gives an overview of the solvable instance sizes of our experiments. The random test instance generator has been implemented to create naturally looking dense or sparse instances by using probabilities that depend on the neighbored fields. Optimal solutions are helpful to evaluate the quality of the solutions of approximation algorithms and heuristics. Knowing how easy different kind of instances can be solved to optimality also helps to predict the difficulty of practical instances that usually have specific patterns.

The concrete structure of this chapter is as follows: In Sec. 4.2 the NP-hardness proofs and also a different representation of the problem is given. In Sec. 4.3 we recap the approximation algorithm for full coverage cycle covers and tours by Arkin et al. [3] and propose

	Solved 50%	Largest solved instance
Full cycle cover	1100-1500	2900*
Full tour	700	2900*
Subset cycle cover	900-1200	1400*
Subset tour	700-900	1400*
Penalty cycle cover	1400*	1400*
Penalty tour	800	1400*

Table 4.2: Optimal solvable 2-dimensional instance size for solving at least 50% of the instances as well as the largest solved instance size with a time limit of 15 minutes. The corresponding experiments have only been performed with turn costs. See Sec. 4.5 for more information. $(*)$ The largest solved instances all have been one of the largest tested instances.

our new method to obtain constant factor approximation algorithms also for the subset and the penalty versions. In Sec. 4.4 we take a look on different integer programming formulations and in Sec. 4.5 experimental evaluations of them are given.

4.1 Problems

In this section, we describe and formalize the different problem variants. We focus on 2-dimensional grid graphs, but the generalization to other grids is rather simple.

Let us begin with the definition of a cycle because it is not necessarily clear.

Definition 1 (Cycle). *A cycle in a 2-dimensional grid graph $G(V, E)$ is a sequence of fields/vertices $v_0, v_1, \ldots, v_n \in V$ with $v_0 = v_n$ and $\forall v_i, v_{i+1} : \{v_i, v_{i+1}\} \in E$. It has length $n \geq 2$ which equals the number of transitions.*

Note that fields can be multiple times in a cycle but there are at least two different fields in a cycle. We can allow cycles with only one field in it, but the meaning of them would be odd so that we prohibit them. However, most of the results in this thesis would actually allow the integration of such cycles.

The cost of a cycle consists of two parts: costs for each turn and costs for each field transition (or its length). We also refer to these costs as turn costs and distance costs. There are three different possible turns: $+90°$, $-90°$, and $180°$. We count $+90°$ and $-90°$ as a single turn. We call $180°$ turns u-turns and count them as two turns. For a hexagonal grid, we would have $\pm 60°$, $\pm 120°$, and $180°$ turns and would count $\pm 60°$ as one turn, $\pm 120°$ as two turns, and $180°$ as three turns. A transition from one field to an adjacent fields adds 1 to the length of the tour. A cycle that only covers two adjacent fields would have 4 turns ($2 \times 180°$) and be of length 2. The cost of a cycle γ with length s and t turns would have a cost $c_{\kappa,\tau}(\gamma) = s * \kappa + t * \tau$ where $\kappa \in \mathbb{R}_0^+$ and $\tau \in \mathbb{R}_0^+$ are two fixed weights. Thus, the cost of a cycle is a linear combination of the amount of turns and its length.

We now consider the details of the different variants.

4.1.1 Full coverage

The simplest problem variants are full coverage cycle cover and tour. In it every field has to be covered by a cycle cover resp. tour. A realistic scenario of such a full coverage tour could be the harvesting of a field and the milling or scanning of an area. For our example with the mosquito hunting drone in the motivation, it could be useful if no expected mosquito distributions are known.

Problem 1 (Full cycle cover). *Given a connected grid graph $G = (V, E)$ and two weights $\kappa, \tau \in \mathbb{R}_0^+$. Find a set of cycles $\mathcal{C} = \{\pi_0, \pi_1, \ldots\}$ such that every vertex $v \in V$ is visited at least once by some cycle of $\mathcal{C}$ and $\sum_{\pi_i \in \mathcal{C}} c_{\kappa,\tau}(\pi_i)$ is minimal.*

A cycle cover is a tour if it only contains a single cycle.

Problem 2 (Full tour). *Given a connected grid graph $G = (V, E)$ and two weights $\kappa, \tau \in \mathbb{R}_0^+$. Find a tour/cycle π in G such that every vertex $v \in V$ is visited at least once and $c_{\kappa,\tau}(\pi)$ is minimal.*

If we only have distance costs (i.e., $\tau = 0$ and $\kappa > 0$), the cycle cover can efficiently be calculated by a 2-factor reduction and the tour variant equals the *traveling salesman problem in grid graphs* for which there is a 6/5-approximation by Arkin, Fekete, and Mitchell [5]. The variant with turn costs has already been considered by Arkin et al. [3, 25], Maurer [7], and de Assis and de Souza [4].

4.1.2 Subset coverage

A practical variation of the previous problem might be that only a subset of the fields has to be visited. In case of our drone example, this might be the fields where the mosquito swarms are expected. All other fields are just feasible positions but without any direct advantages (except for possibly providing a short way to another important position). This problem variation is similar to the *Steiner tree on graphs* as a variation of the *minimum spanning tree*; see the book of Hwang et al. [66] for an overview of the Steiner tree problem.

Problem 3 (Subset cycle cover). *Given a connected grid graph $G = (V, E)$, a set $S \subseteq V$, and two weights $\kappa, \tau \in \mathbb{R}_0^+$. Find a set of cycles $\mathcal{C} = \{\pi_0, \pi_1, \ldots\}$ such that every vertex $v \in S$ is visited at least once by some cycle of $\mathcal{C}$ and $\sum_{\pi_i \in \mathcal{C}} c_{\kappa,\tau}(\pi_i)$ is minimal.*

And the respective tour version of the problem:

Problem 4 (Subset tour). *Given a connected grid graph $G = (V, E)$, a set $S \subseteq V$, and two weights $\kappa, \tau \in \mathbb{R}_0^+$. Find a tour/cycle π in G such that every vertex $v \in S$ is visited at least once and $c_{\kappa,\tau}(\pi)$ is minimal.*

Obviously, Problem 3 and Problem 4 are equivalent to Problem 1 resp. Problem 2 if $V = S$.

4.1.3 Penalty coverage

In the case of the example with the drone, the previous variation allowed us to only enforce the lucrative fields to be covered. A natural extension of this is to not to force visiting

lucrative fields but to give a prize for it. This can also be seen as a penalty for not visiting these fields. Some fields might be lucrative but very far away from the other lucrative fields and an integration of them might not be reasonable. This new problem formulation allows the tour to actually skip these fields with a bad usage/cost-ratio while the previous formulation would have enforced an additional expensive preoptimization which is also inaccurate because the costs depend on the actual tour.

This problem variation is analog to the *penalty traveling salesman problem (PTSP)* variation of the TSP. For the PTSP, a 2-approximation is known for the case that the triangulation inequality holds for the corresponding graph [53]. The PTSP is often also wrongly called the *prize-collecting traveling salesman problem (PCTSP)* but the PCTSP also contains a quota that has to be fulfilled, see [54]. For the PCTSP only a $O(\log^2(\min(Q,n)))$-approximation algorithm by Awerbuch et al. [67] is known to us where Q is the quota. PTSP and PCTSP both have only distance costs and no turn costs.

The definitions of our cycle cover and tour variant can be given as follows:

Problem 5 (Penatly cycle cover)**.** *Given a connected grid graph $G = (V,E)$, a penalty function $\rho : V \to \mathbb{R}_0^+$, and two weights $\kappa, \tau \in \mathbb{R}_0^+$. Find a set of cycles $\mathcal{C} = \{\pi_0, \pi_1, \ldots\}$ such $\sum_{\pi_i \in \mathcal{C}} c_{\kappa,\tau}(\pi_i) + \sum_{v \in V \setminus V(\mathcal{C})} \rho(v)$ is minimal. $V(\mathcal{C})$ denotes the fields covered by $\mathcal{C}$.*

Problem 6 (Penalty tour)**.** *Given a connected grid graph $G = (V,E)$, a penalty function $\rho : V \to \mathbb{R}_0^+$, and two weights $\kappa, \tau \in \mathbb{R}_0^+$. Find a tour/cycle π in G such that $c_{\kappa,\tau}(\pi) + \sum_{v \in V \setminus V(\pi)} \rho(v)$ is minimal.*

Often the PTSP is also defined with a root vertex that has to be covered. This could be the start point of the drone in our example. However, this is actually also contained in our formulation as we can simply set the penalty for this root field very high.

Obviously, the penalty coverage problem can model the subset coverage problem by using a high penalty for the fields in the subset. This high penalty could, e.g., be the cost of a greedily calculated tour.

4.2 Hardness

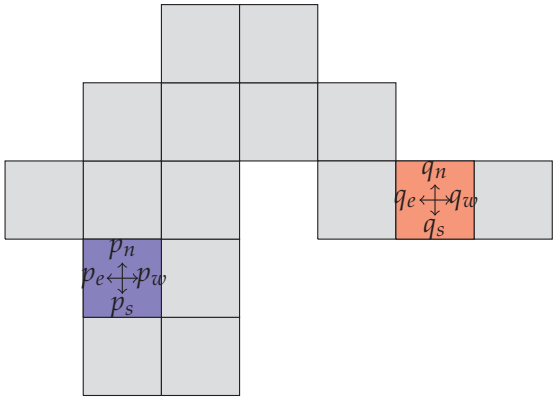
Arkin et al. [3] showed that it is NP-hard to find a tour with a minimum number of turns in 2-dimensional grid graphs even if the grid graph is thin and a cycle cover with minimal turns is given. This also implies the NP-hardness of the tour versions of the other problem variations considered in this thesis. Whether the problem of finding a cycle cover with minimal turns is also NP-hard remained an open problem until now. It is listed as *Problem* 53 in *The Open Problems Project* edited by Demaine, Mitchell, and O'Rourke [2]. In case it would have been polynomial solvable, the approximation factor of the approximation algorithm for the tour problem by Arkin et al. [3] (see also Sec. 4.3.2) would have improved from 3.75 to 1.5 (without distance costs).

In Sec. 4.2.2, we prove that it is NP-hard to find a cycle cover with minimum turn costs in 2-dimensional grid graphs (Problem 1), answering the above mentioned open problem. Note that this is the simplest variant (regarding underlying graph and constraints) and the

hardness of all other considered cycle cover problem variants follows trivially from it. For the variation of the problem where only a subset has to be covered (Problem 3), we give a significantly simpler proof in Sec. 4.2.3 that is similar to the NP-hardness proof of the angular metric cycle cover problem by Aggarwal et al. [1]. It is not only simpler than the hardness proof for full coverage cycle cover but it also works on thin 2-dimensional grid graphs. For thin 2-dimensional grid graphs, the full coverage cycle cover problem can be solved in polynomial time as shown by Arkin et al. [25]. However, first we consider full coverage cycle cover in an alternative representation which is not only helpful to improve the understanding of the problem but is also used multiple times in this thesis.

4.2.1 Alternative problem representation

A problem with the grid graph representation of the problem instance and its solution is that vertices (fields) and edges often have to be visited multiple times. This sometimes makes the arguing much harder because only one visitation of a field is to satisfy the constraint and the others are only auxiliary to connect to non-adjacent fields. Alternatively, we could connect these two fields directly by an edge that possibly represents a path over multiple fields and possibly even contains turns. The cost of a such a path depends on the direction the source field has been entered. Hence, there are 16 different edges between any two fields (one for every direction combination) as can be seen in Fig. 4.2. We can represent each enter/leave direction by a subvertex and enforce that a field is always left through the subvertex opposite to the subvertex it is entered. The costs for internal turns can simply be added to the edges. The problem of finding a cycle cover now becomes the problem of finding a set of edges such that for every field either the north and south or the east and west subvertex each have exactly one edge.



	q_n	q_e	q_s	q_w
p_n	4	3	4	5
p_e	5	4	5	6
p_s	6	5	6	7
p_w	5	4	5	6

Figure 4.2: Instead of considering only edges between two adjacent fields we can build a graph with the shortest connection between any two fields. Every field has four directions which are represented by vertices. If a field is entered through the north vertex, it has to leave through the south vertex.

Theorem 1. *An optimal solution in the alternative representation can be efficiently mapped onto an optimal solution of the corresponding cycle cover problem.*

Proof. It is easy to see that every solution of the alternative representation can be mapped onto a solution of the corresponding cycle cover problem with same cost. For this, every edge is simply replaced by the sequence of implicitly visited fields. We now show that we can also obtain a solution for the alternative representation from an optimal solution of the cycle cover problem with the same cost. For every field we have to decide in which cycle it is explicitly such that every cycle has at least two explicit fields. For every cycle we first choose its turn fields (at least two) to be explicit and all other fields can be chosen arbitrarily. In case a field contains a turn for two (or more) different cycles, we can do a local modification that either removes a turn or connects the two cycles without increasing the costs. This local modification is a simple case distinction and left to the reader. □

We can use the alternative representation to get a simple result that helps us to understand what the difficult parts in a cycle cover are.

Theorem 2. *If we know a superset V'_H of all fields V_H that are only passed straight horizontally and a disjunct superset V'_V of those fields V_V that are only passed straight vertically, then we can calculate an optimal cycle cover in polynomial time.*

Proof. The primary observation for this result is that for a field that has a turn at the position in the cycle it explicitly is in, we can simply replace the two used edges from v_n and v_s with edges from v_e and v_w and vice versa without a change in the costs. These are the only important fields for the previous theorem and we did not change their explicit positions. For a field passed straight in two orientations, we can simply exchange the position it is explicitly in. Now only the fields in V_H and V_V remain and for those we already now the used vertex pair. Hence, we can choose an arbitrary vertex pair for all fields in $V \setminus (V'_H \cup V'_V)$ and for V'_H we choose the horizontal pair and for V'_V respectively the vertical pair. For V_H and V_V the right pairs are automatically chosen and those fields $V'_H \setminus V_H$ and $V'_V \setminus V_V$ belong to the previous category for which the chosen pair does not matter. Now the problem is a simple minimum weight perfect matching. □

This means that the difficult part is to determine the passing orientation of these exclusively straight passed fields which is somehow counterintuitive because, e.g., a strip cover (polynomial time for 2-dimensional grid graphs) gives a good hint and we are allowed to guess wrong for all other fields.

4.2.2 Cycle cover with turn costs is NP-hard

In this section, we prove that full cycle cover in 2-dimensional grid graphs with turn costs (distance costs are not needed) is NP-hard. This answers the open *Problem* 53 in *The Open Problems Project* edited by Demaine, Mitchell, and O'Rourke [2]. The hardness of the problem is not obvious: usually large parts of a solution can be easily deduced by local information and 2-factor techniques (see Theorem 2). A further problem is that there is an

interference by neighbored fields that have to be covered too. In a variation of the problem where we only need to cover a subset of fields (see Problem 3), we can easily eliminate the interference and obtain a significantly simpler proof (see Sec. 4.2.3).

Theorem 3. *It is NP-hard to find a cycle cover with a minimum number of turns in a 2-dimensional grid graph (Problem 1 with $\tau = 1$ and $\kappa = 0$).*

Proof. Following pages. □

The proof is structured as follows: First, we give an example of the reduction to provide a first impression of the general idea before we describe the construction and try to give an intuition of the correctness. Afterwards, we give a lower bound on the necessary turns and show that this lower bound can be matched by a satisfiable formula. If a formula is not satisfiable, we show that number and positions of turns implied by the lower bound are not sufficient for a full coverage.

In this proof, we reduce the problem to the *One-In-Three-3SAT* problem, which has been proven to be NP-hard by Schaefer [68]. In *One-In-Three-3SAT* we have to decide whether a formula in conjunctive normal form with only three literals per clause can be satisfied with only exactly one literal per clause becoming true (hence, exactly two literals being false). For example, $(x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$ is not satisfiable whereas $(x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4})$ is satisfiable.

Example

In Fig. 4.3 we see the construction and its solution for the formula $(x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4})$. The formula itself has the unique satisfiable assignment of x_2 and x_4 being the only variables set to true. For each variable a blue box is visible that stretches horizontally through the construction and the two ends of the contained construction are connected by a slim path of fields highlighted in purple. This path enforces its covering cycle to go through the box of the variable since a double coverage of itself would be to expensive due to the wavy part. It can be seen that this covering cycle takes the upper way if the corresponding variable is set to true and takes the lower way if the variable is set to false (see also Fig. 4.4). There are also two red boxes, one for each clause, that stretch vertically through the construction. They each contain 3×3 'connectors' to the contained variables. Each connector contains a red highlighted field that is difficult to cover. There are always three connectors connected vertically and the logical expression of the left most would for example be $x_1 \leftrightarrow \overline{x_2} \wedge \overline{x_3}$. Combined with the next two vertical connector triples, they form the actual clause. There are a lot of simple crossings of paths with width of two fields but they do not matter because a 'usage' of these crossings would require additional turns (exceeding the lower bound). If they are used in an optimal solution, the corresponding formula cannot be satisfiable.

Construction

While the example gave a top down view, we now consider the details of the construction in a bottom up view. As a formal description of the construction would probably be

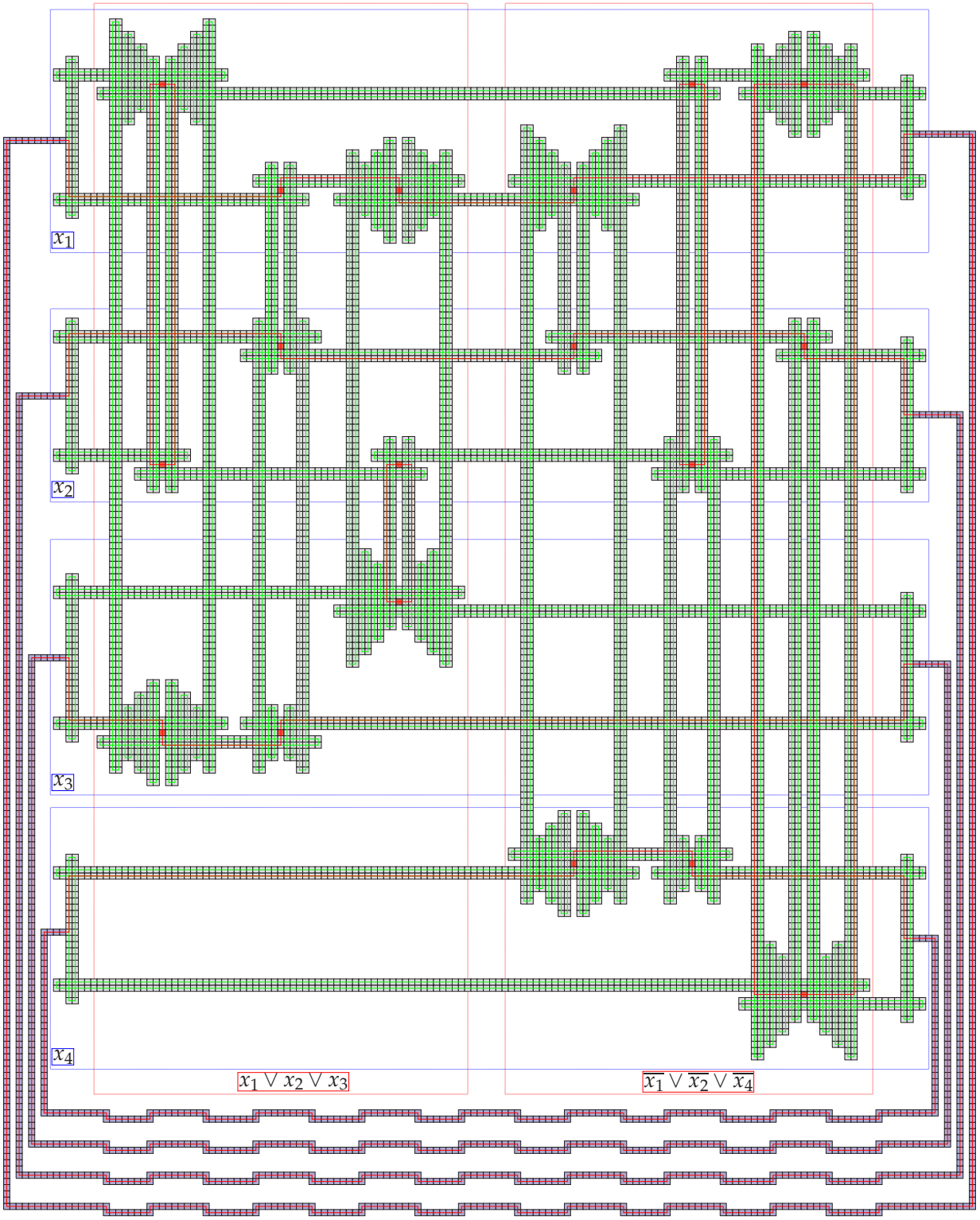


Figure 4.3: The construction for the One-In-Three-3SAT formula $(x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4})$. The provided solution is $x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 1$. In case the resolution of your copy is too low, you can find a scalable version on `http://krupke.cc/m/tubs/mt/npexample.pdf`

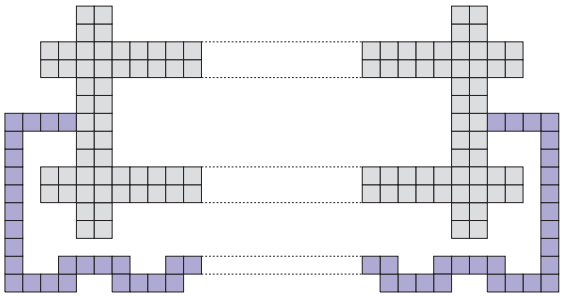


Figure 4.4: The raw construction for a variable. The purple part has to be covered by letting the cycle go through the gray part. If it goes through the upper way, the variable is set to true. If it goes through the lower way, the variable is set to false.

unreadable, we only give an informal description that mainly focuses on the idea and is based on the actual train of thoughts that led to the proof.

We know by Theorem 2 that useful information can only be hidden in fields for which we do not know in what orientation they are exclusively passed straight. This means that, e.g., fields on the boundary are not of interest. Also, fields where we know that there is a turn or that are crossed straight in both orientations are irrelevant. They might only be used indirectly for information propagation.

Based on these observations, we consider variants of the construction in Fig. 4.5 as fundamental information element.

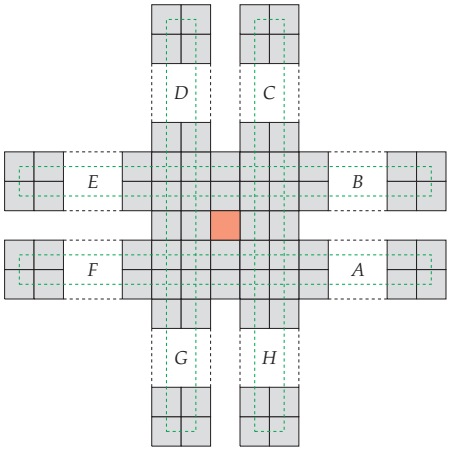
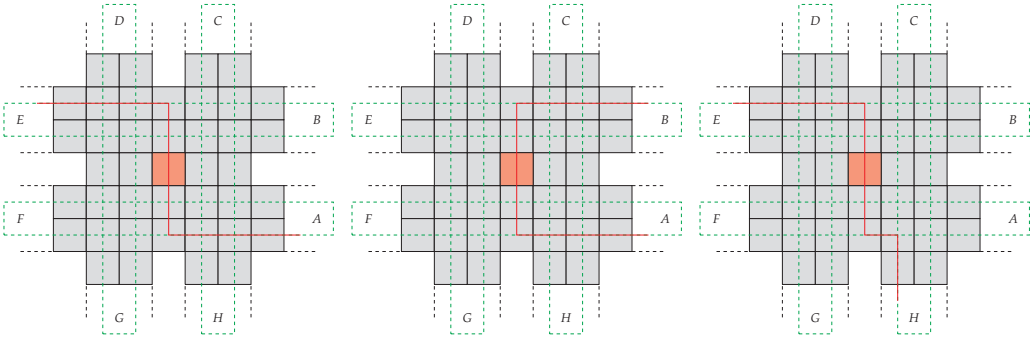


Figure 4.5: Basic block for hardness proof. Only the red field is of interest as described in the text.

The optimality of the green cycles is obvious, and thus the corresponding fields are trivially covered. Covering the red field is more difficult. Covering it by an additional cycle or reforming the green cycles costs always at least 4 turns. Assume we create a path from one entry (A, B, C, D, E, F, G, H) to another and cover the red field by this path. For example, a path entering at A and leaving at E would allow us to cover the red field at a cost of 2 while a path entering at A and leaving at D has at least 3 turns if also covering the red field, see Fig. 4.6. Actually, we can cover the red field with a cost of 2 exactly with



(a) S-Turn (b) U-Turn (c) Bad turn

Figure 4.6: Three ways of covering the red field. The first two keep the orientation and have a cost of 2. The third one changes the orientation, and thus has a cost of at least 3.

the s- and u-turns, i.e., paths that do not change the orientation. Further, it can be seen that the red field always needs at least two turns within this construction.

Now consider the following construction in Fig. 4.7 that uses three variations of the element above. Assume all turns made outside of the construction are free and all six 'exits' A, B, C, D, E, F are connected from outside. This means that every red highlighted field can be covered by a cycle of cost two that goes through the exits above and below (see blue paths in Fig. 4.7). The green cycles are still necessary and there is no use in using them for covering the red field, hence we can concentrate on how to cover the red highlighted fields by additional cycles. It can be seen that there are only five potential cycles that only involve a cost of two turns per red field: The three mentioned blue cycles that use the exterior and the two interior cycles marked in red in Fig. 4.7 (the blue cycles can slightly be shifted but this doesn't change them usefully). This results in only three potential optimal solutions:

1. Using the left blue cycle to cover the left red field and the right red cycle to cover the middle red field and right red field.
2. Using the right blue cycle to cover the right red field and the left red cycle to cover the middle red field and the left red field.
3. Using all three blue cycles to cover each red field separately.

Let us assign a boolean variable for the usage of each blue cycle: x_l for using the left blue cycle, x_m for using the middle blue cycle, and x_r for using the right blue cycle. The optimal solutions then result in the following formula: $x_m \leftrightarrow x_l \wedge x_r$.

In Fig. 4.8 we have an alternative but analog construction that results in the formula $x_l \leftrightarrow x_m \wedge x_r$. The construction can be simply mirrored to obtain the formula $x_r \leftrightarrow x_m \wedge x_l$.

We can combine three of these constructions for six such exterior cycles, hence six variables, to a construction as shown in Fig. 4.9. This results in the formula

$$(y_1 \leftrightarrow y_4 \wedge y_6) \wedge (y_3 \leftrightarrow y_2 \wedge y_6) \wedge (y_5 \leftrightarrow y_2 \wedge y_4)$$

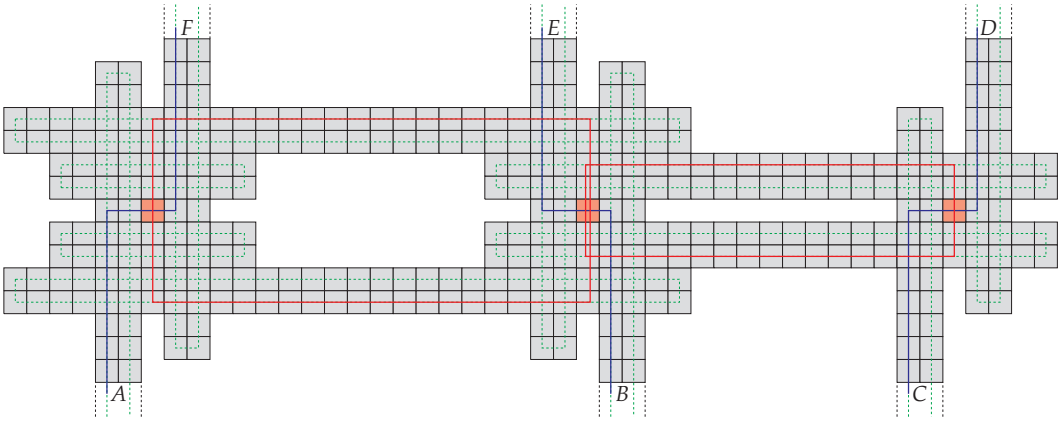


Figure 4.7: Assuming that all turns made outside are free, there are only five potential cycles to cover the red fields: The three exterior blue cycles and the two interior red cycles. Assigning the usage of the left blue cycle the boolean variable x_l, analogous for the middle and the right x_m resp. x_r leads to the boolean formula $x_m \leftrightarrow x_l \wedge x_r$.

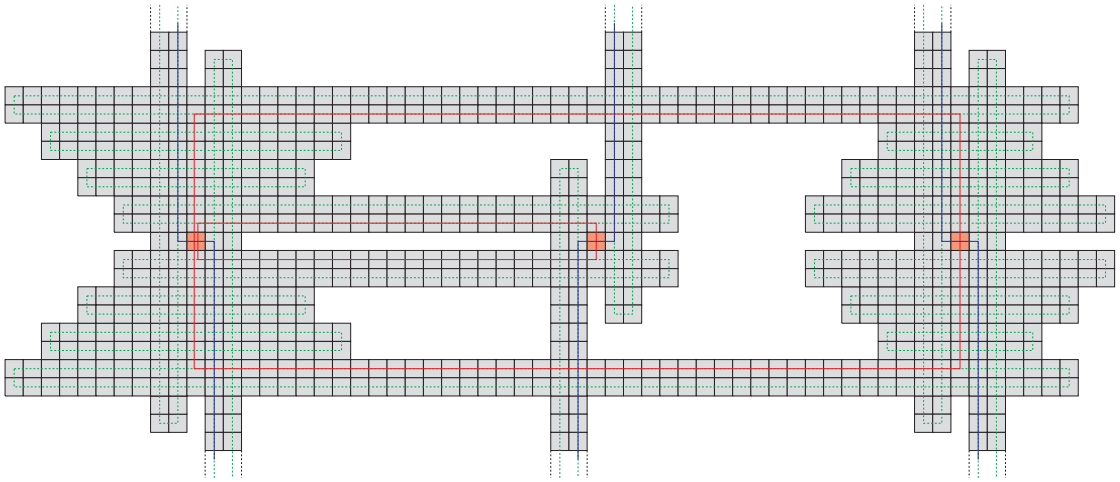


Figure 4.8: Analogous to Fig. 4.8 but leading to the formula $x_l \leftrightarrow x_m \wedge x_r$. $x_r \leftrightarrow x_m \wedge x_l$ can be obtained by simple mirroring.

Let us now assume that we externally enforced $y_1 \leftrightarrow \overline{y_2}$, $y_3 \leftrightarrow \overline{y_4}$, and $y_5 \leftrightarrow \overline{y_6}$. Thus, we could substitute y_1 by x_1, y_2 by $\overline{x_1}$, y_3 by x_2, y_3 by x_2, y_4 by $\overline{x_2}$, y_5 by x_3, and y_6 by $\overline{x_3}$. This results in the formula

$$(x_1 \leftrightarrow \overline{x_2} \wedge \overline{x_3}) \wedge (x_2 \leftrightarrow \overline{x_1} \wedge \overline{x_3}) \wedge (x_3 \leftrightarrow \overline{x_1} \wedge \overline{x_3})$$

or stated differently

$$x_1 + x_2 + x_3 = 1$$

Hence, we have our first *One-In-Three-3SAT* clause $(x_1 \vee x_2 \vee x_3)$. It can easily be seen that by swapping the height of, e.g., x_1 and $\overline{x_1}$ we can obtain the *One-In-Three-3SAT* clause $(\overline{x_1} \vee x_2 \vee x_3)$. On this way we can construct all necessary clauses.

The enforcement of $y_1 \leftrightarrow \overline{y_2}$, $y_3 \leftrightarrow \overline{y_4}$, and $y_5 \leftrightarrow \overline{y_6}$ is done by the variable constructions as displayed in Fig. 4.4. Thus, the exterior cycles are the cycles that also cover the purple highlighted paths. The usage of them for another variable needs additional turns.

By concatenation of multiple such clauses we can construct any *One-In-Three-3SAT* formula as we saw in Fig. 4.3.

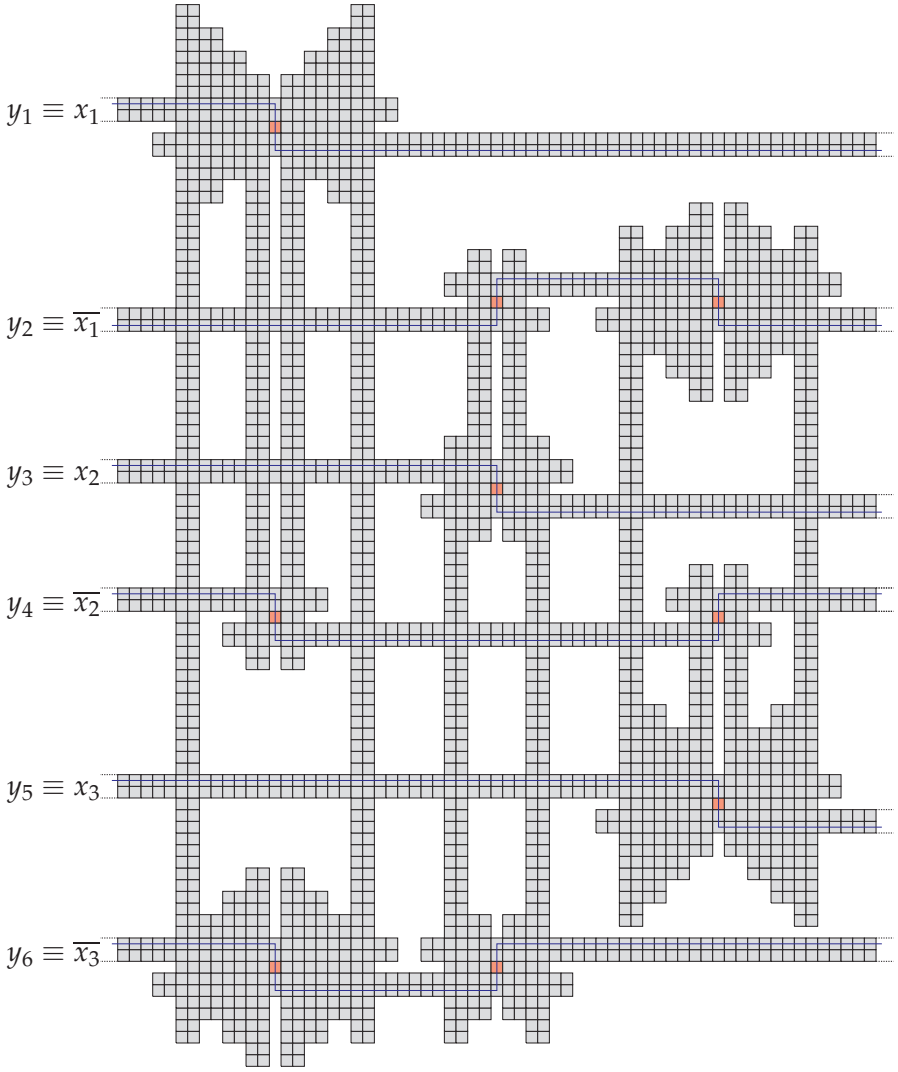


Figure 4.9: Combining three constructions as in Fig. 4.7 and Fig. 4.8 to obtain a *One-In-Three-3SAT* clause.

Tight lower bound

We now give a tight lower bound one the number of turns in a cycle cover. In order to achieve it, the turns have to be made at very specific positions. Fortunately, it is sufficient to limit our view onto local components and state that in each of these disjunct components a specific number of turns has to be made. The global lower bound is then the sum of all local lower bounds. Later we show that these turns are only sufficient if the

corresponding formula is satisfiable.

We first provide a trivial but important observation and two simple lemmas that help us to deduce necessary turn positions.

Observation 1. *There has to be a turn on every corner field.*

Lemma 1. *On every full strip in the grid graph, a cycle cover has to have an even number of turns (u-turns counting twice).*

Proof. For every turn that enters the strip, there needs to be a turn for leaving it. Turns that do not enter/leave the strip are u-turns. □

Lemma 2. *Given any field f of a grid graph and a corresponding cycle cover C. If there is no turn on f, then*

- *there has to be a turn left of f and another turn right of f (on the horizontal full strip through f), or*
- *there has to be a turn above of f and another below of f (on the vertical full strip through f).*

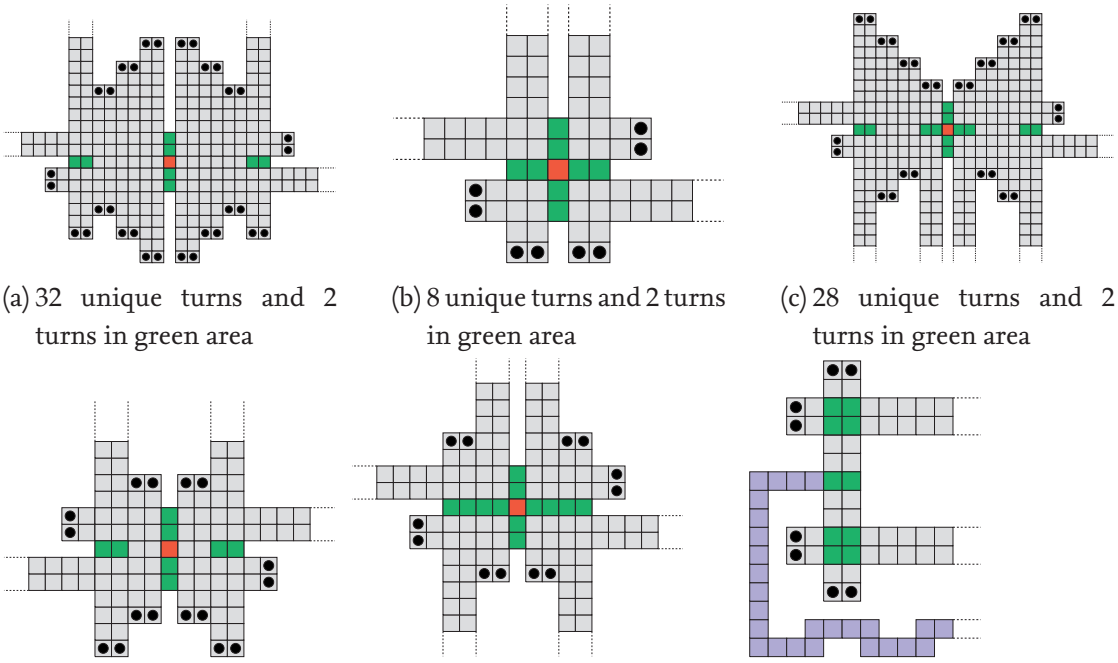
If there is a turn on f, there are at least three turns on the horizontal and vertical full strip (star) through f.

Proof. In case there is no turn on f, there is a straight part of a cycle going through it. This straight part needs to turn at some time in both directions in order to close the cycle. If there is a turn on f, the claim follows by Lemma 1. □

Now we can deduce the turn positions for all essential parts of the construction in the next lemma.

Lemma 3. *In Fig.4.10, the essential parts of the construction are displayed (possibly used mirrored). For these parts we can give the following lower bounds on the necessary number of interior turns and constraints on the positions of turns if this bound is tight:*

- *In a part as depicted in Fig. 4.10a, we need at least 34 interior turns. To achieve exactly this lower bound, there has to be exactly one turn at every black dot and the remaining two turns have to be in the green area.*
- *In a part as depicted in Fig. 4.10b, we need at least 10 interior turns. To achieve exactly this lower bound, there has to be exactly one turn at every black dot and the remaining two turns have to be in the green area.*
- *In a part as depicted in Fig. 4.10c, we need at least 30 interior turns. To achieve exactly this lower bound, there has to be exactly one turn at every black dot and the remaining two turns have to be in the green area.*
- *In a part as depicted in Fig. 4.10d, we need at least 18 interior turns. To achieve exactly this lower bound, there has to be exactly one turn at every black dot and the remaining two turns have to be in the green area.*



(a) 32 unique turns and 2 turns in green area

(b) 8 unique turns and 2 turns in green area

(c) 28 unique turns and 2 turns in green area

(d) 16 unique turns and 2 turns in green area

(e) 12 unique turns and 2 turns in green area

(f) 8 unique turns and 2 turns in green area.

Figure 4.10: Lower bounds on the number of turns in the different parts of the construction (might appear mirrored). The black dots represent turn position necessary to achieve exactly the lower bound.

- *In a part as depicted in Fig. 4.10e, we need at least 14 interior turns. To achieve exactly this lower bound, there has to be exactly one turn at every black dot and the remaining two turns have to be in the green area.*
- *In a part as depicted in Fig. 4.10f, (excluding the purple area) we need at least 10 interior turns. To achieve exactly this lower bound, there has to be exactly one turn at every black dot and the remaining two turns have to be in the green area.*

Proof. It is obvious that there needs to be at least one turn in every corner field (Observation 1). For the other fields, we need some simple additional arguments based on Lemma 1 and 2.

We give the details for the part in Fig. 4.10c and leave the others to the reader as the adaption is straightforward and would only waste space. The used arguments are highlighted in Fig. 4.11 . First, there are 16 corner fields marked by black circles with at least one turn each, leaving us 14 turns. Second, to fulfill Lemma 1 there has to be an additional turn on each yellow strip. On the two orange strips there have to be two further turns two cover each of the fields with a black square. With the two further turns needed on the green highlighted fields to cover the red field, we are out of 'independent' turns. We now select fields for applying Lemma 2 (marked by hollow circles). Because our remaining

turns are limited to the colored fields (with corresponding number per area), the claim follows straightforward. Lemma 2 could in this case also be simplified to 'there has to be a turn in a colored field above and below each hollow circle'.

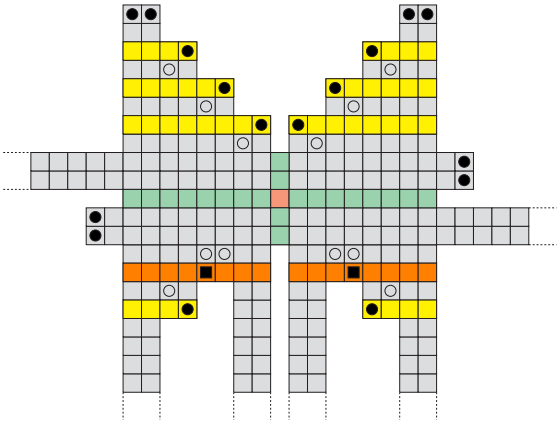


Figure 4.11: Auxiliary figure for proof of Lemma 3.

□

We now can state the lower bound:

Theorem 4. *Given a construction, as described above, for an arbitrary formula with v variables and c clauses. Let k be the number of purple highlighted corner fields. Then every cycle cover has to have at least $20 * v + 190 * c + k$ turns.*

Proof. Sum up the local bounds for all local parts using Lemma 3. There is no interference between these local bounds since we did not charge for exterior (not inside considered local part) turns made by possible cycles in the proof. □

Upper bound for satisfiable formulas

We can construct a solution that matches the lower bound given in Theorem 4 in case the corresponding formula of the grid graph construction is satisfiable.

Theorem 5. *Given a construction, as described above, for an arbitrary formula with v variables and c clauses. Let k be the number of purple highlighted corner fields. If the formula is satisfiable, there exists a cycle cover with $20 * v + 190 * c + k$ turns.*

Proof. This is easily extractable from the construction description and Fig. 4.3. Hence, we only repeat the rough details: Cover the gray fields only using simple 4-turn cycles on the explicit turns (black dots) like in Fig. 4.10. Select an arbitrary but fixed satisfiable solution for the formula. If a variable assignment is true, add a cycle that goes with a minimum number of turns through all purple fields of the variable and all upper red fields of the variable. This cycle has a turn at every purple corner field belonging to the variable, two turns per red field, and two further at each end to connect to the purple fields. If a variable assignment is false, do this analogously but for the lower row of red fields belonging to

this variable. Now, there are four remaining red fields per clause which we can cover by two simple 4-turn cycles because the clause is satisfied by exactly one term. As Lemma 3 left us with exactly 2 turns to select per red field, this matches the lower bound. □

Lower Bound for unsatisfiable formulas

In case the corresponding formula of the grid graph construction is not satisfiable we can deduce that the number of turns given by Theorem 4 is not sufficient for a full coverage. We first show that in a lower bound matching cycle cover, we can separate a cycle cover for the gray fields from the rest.

Lemma 4. *Given an optimal cycle cover that matches the lower bound of Theorem 4, hence fulfills the restriction on the position of turns as in Lemma 3. Then we can modify the solution such that the gray fields are covered by a set of cycles that uses exactly the black dotted fields in Fig. 4.10.*

Proof. Due to the strict limitations of Lemma 3, the only alternative turns are in the green areas. However, there is first the restriction that we can only use one turn in one coherent green area (Lemma 2) and further every turn on a green field needs a horizontal and a vertical partner turn on a green field to fulfill Lemma 1. This results in two turns in green fields only being encapsulated by two turns on black dotted fields. As the reachable possible turns for a black dotted field are strictly limited, we only have the simple case (two green fields between two black dots) as in Fig. 4.12 for which we can simply split the cycle into two. This is sufficient since for a cycle to have turns on the green fields and the black dots there has to be a subsequent 'green turn' after a 'black dot turn' which we, however, can all eliminate by the previous argument. Note that the part as in Fig. 4.10f is slightly different with respect to Lemma 1 due to an additional turn in the purple area but the argument works analogously. □

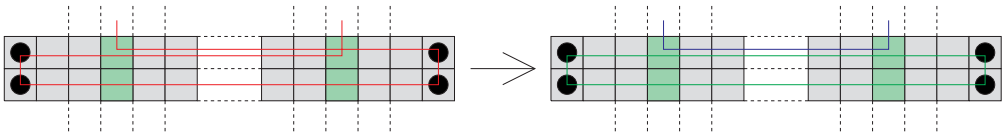


Figure 4.12: Lemma 1 enforces the turns on the green fields to be on the same height. The cycle part can be easily separated to a cycle that covers the gray fields and a connection between the turns in the green fields.

This allows us to only concentrate on covering the red and the purple fields because all gray fields are already covered by the 'black dot cycles'. We now show that every cycle cover that matches the lower bound results in a variable assignment (the cycle that covers the purple fields either selects the 'true' line or the 'false' line and does not switch in between).

Lemma 5. *In a cycle cover that matches the lower bound of Theorem 4, the cycle that covers the purple fields of a variable also covers either all red fields belonging to a true assignment of this variable and no other red field or all red fields belonging to a false assignment of this variable and no other red field.*

Proof. It is unreasonable (and impossible without exceeding the lower bound because we only have free turns in the green fields) to cover any purple field twice, hence the 'variable cycle' has to go through the gray and red fields. We are only allowed to make turns in the green fields as marked in Fig. 4.10 and only at most one in every connected green area. The variable cycle can go up or down at the beginning (Fig. 4.10f) and then has to stick with that decision us there is only one possible 'partner turn' to achieve an even number of turns on every strip. The 'partner turn' is the subsequent turn of the cycle after applying Lemma 4 as it is the only choice. This enforces us to go through until the end and then close the cycle. This can also be seen simplified in Fig. 4.13. □

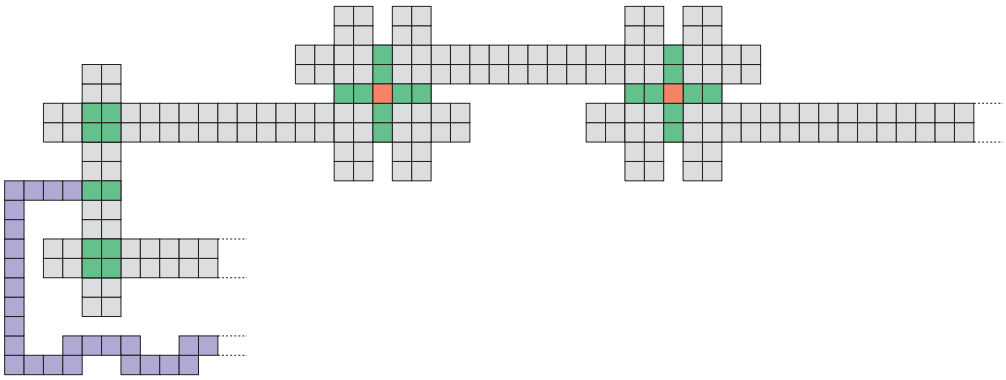


Figure 4.13: If the cycle goes up, Lemma 1 enforces us to cover exactly all red fields at this height as we have only one choice for the position of the 'partner turn'.

Now only the red fields not covered by the variable assignment cycles are left. We have only two turns for each of them, hence we need at least two still uncovered red fields in any additional cycle and this cycle has to have at most twice the number of turns than the number of its newly covered red fields. This, however, is impossible for unsatisfied clauses and we must exceed the lower bound.

Theorem 6. *Given a 2-dimensional grid graph produced by the above procedure for an arbitrary One-In-Three-3SAT formula. Let C be a set of cycles with minimum turn costs. Let v be the number of variables in the corresponding formula, c the number of clauses, and k be the number of purple corner fields. If the formula is not satisfiable, the cycle cover has to have at least* $20 * v + 190 * c + k + 1$ *turns.*

Proof. Assume there exists a cycle cover with only $20 * v + 190 * c + k$ turns (matching the lower bound of Theorem 4). We now show that this number of turns cannot suffice to cover all fields. By separating the cycles on the black dotted fields using Lemma 4, only $4 * v + 18 * c + k$ turns remain for the red and purple fields.

In Sec. 4.2.1 we considered an alternative problem representation where we have direct paths between any two fields and not vertex in the representation is visited or edge used twice. We can apply this representation on the red and purple fields, and thus do no longer need to care for the gray fields. We now analyze the costs of the edges in this representation

and will see that only a very limited set of edges can be used if the cost shall not exceed $4 * v + 18 * c + k$ turns. First, we limit the set of interesting vertices to the red fields and the purple end fields.

A simple observation is that the purple fields per variable are each covered by coherent paths, see Fig. 4.14. It is not reasonable to visit any field of it twice. We thus can assume the two end fields for each variable to be connected, and thus replace the other purple fields by an edge between the end fields that has to be used. Hence, for each purple end field only one additional edge has to be selected. The cost of these edges is exactly k such that we are only allowed to add edges in worth of $4 * v + 18 * c$.

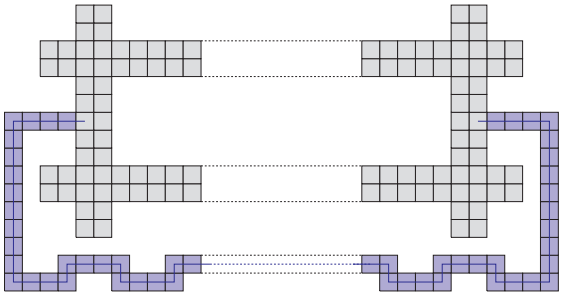
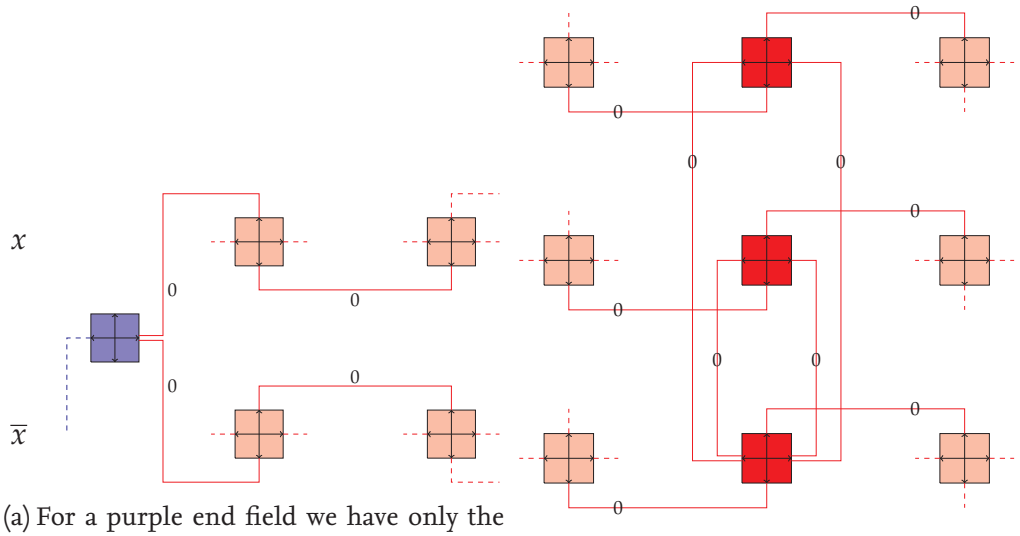


Figure 4.14: In any optimal solution, the purple fields are covered by a single cycle that goes once through the purple fields and connects over the gray fields.

This value could alternatively also be described as $2 * b + 2 * r$ where b is the number of purple end fields and r the number of red fields. As every purple field has to have one additional outgoing edge and every red field has to have two additional outgoing edges, we can simply subtract one from the costs of each edge adjacent to a red field and two from every field adjacent to a purple field. The only edges that any solution with cost no more than $4 * v + 18 * c + k$ is allowed to select are those edges with cost zero in the modified graph.

For any purple end field there is only the next red field for the false assignment of its variable and the next red field for the true assignment of its variable (see Fig.4.15a). Further, if we have decided for the true or false assignment we have to stick with this decision because the only outgoing edge with zero cost from this incoming edge is the edge to the next red field of the assignment (see Fig. 4.15b). This propagates until we reach the purple end field of the other side. This has also been shown in Lemma 5 using a different argument.

The remaining red fields are not able to chose the edges to the neighbored red fields of the same variable assignment as they would get stuck at the last red fields due to the same propagation mentioned above (see Fig. 4.15a). They can only connect to the two fields above or below depending on the type of triple. Hence, we have to use the red cycle as described in the construction. These triples had a logic matching to the formula in order to allow this. As a not satisfying variable assignment violates at least one of these logic triples, for at least one red field we need more than two turns resulting in a violation of the lower bound. □



(a) For a purple end field we have only the possibility of taking the next red field of the true or false assignment.

(b) Once going sidewards, we have to go sidewards until the end.

Figure 4.15: Graph abstraction

NP-hardness

Finally, we can state the main theorem that the bound on the number of turns can only be fulfilled if the formula is satisfiable.

Theorem 7. *Given a construction, as described above, for an arbitrary formula with v variables and c clauses. Let k be the number of purple highlighted corner fields. Then the formula is satisfiable if and only if there exists a cycle cover with $20 * v + 190 * c + k$ turns.*

Proof. Theorem 5 and Theorem 6. □

This concludes the proof of the NP-hardness of full coverage cycle cover (Theorem 3).

4.2.3 A simple NP-hardness proof for subset cycle cover

One thing that made the NP-hardness proof of cycle cover difficult was that the coverage of the auxiliary/connection fields interfered with the coverage of the 'logic fields' because these fields have to be covered even if we do not use the connection. In subset cycle cover, we can eliminate this interference by only demanding the coverage of the 'logic fields'. This allows us not only to provide a simpler proof for subset cycle cover but also to show that subset cycle cover is already hard in thin grid graphs. This is worth mentioning as full coverage cycle cover in thin grid graphs can be solved in polynomial time (see [3]).

The core idea of the reduction equals the NP-hardness proof for the angular metric cycle cover problem by Aggarwal et al. [1] but the construction is simpler because the grid graph allows us to easily limit the possibilities.

Theorem 8. *The subset cycle cover problem is NP-hard even in thin graphs.*

Proof. Due to the simplicity of the reduction, we keep the proof rather informal by using the example illustrated in Fig. 4.16. We only use turn costs and set the distance costs to zero.

Every variable consists of a U-form with two lines connecting the ends of the U. The bottom of the U has to be covered anyways and we are free to also either cover the upper or the lower line without additional turn costs. Covering the upper line means setting the variable to true and covering the lower line setting the variables to false. These two lines intersect with the constructions for the clauses. Every clause construction consists of three vertical lines (intersecting with the variable constructions) and a horizontal connection at each end. At the top and the bottom are each a field that has to be covered. The cheapest way to cover these two fields is by a cycle that goes through the vertical lines. Thus, we are able to cover two of three vertical lines per clause construction for free. The two covered lines represent the two terms of the clause that have to be false.

We now mark additional fields of the intersection of the variable constructions and the clause constructions to also be in the subset of fields to be covered. This is done in a way such that if we set a variable to a specific value for all clauses that now become true the two lines to be covered are automatically enforced. In the example, if we set x_1 to true and cover the upper line, we have to use the two lines on the right in clause $x_1 \vee x_2 \vee x_3$ because otherwise the two lower points in the intersection are not covered. This on the other hand enforces x_2 and x_3 to be false as the corresponding two left fields to be covered can no longer be covered by the clause's cycle. If and only if we do not need any additional cycles, the cycle cover represents a valid solution for the *One-In-Three-3SAT* formula.

□

4.3 Approximation algorithms

In the previous section, we have shown that even the simplest problem variants are already NP-hard, and thus there cannot exist an optimal algorithm with polynomial runtime for any of the considered problems (unless $P = NP$). In this section, we provide constant factor approximation algorithms for all problem variants. We start by proposing a technique, the *Match-Max-Pairs-Algorithm*, for the approximation of an artificial problem that is used as a base component for the approximation algorithms of our problems. For the full coverage variants, we recap the minimum strip cover technique of Arkin et al. [3] which is better than our new technique in 2-dimensional grid graphs but is less general and only works for full coverage in grid graphs. Already for 3-dimensional grid graphs or hexagonal grid graphs, our method is superior regarding the proven approximation factor. Our new technique, in contrast, not only also works for subset and penalty coverage but even for a more generic geometric problem formulation (Chapter 5).

4.3.1 Match-Max-Pairs-Algorithm

In this section, we state an artificial problem and provide an approximation algorithm for it. This algorithm later is used to obtain cycle covers for the actually considered prob-

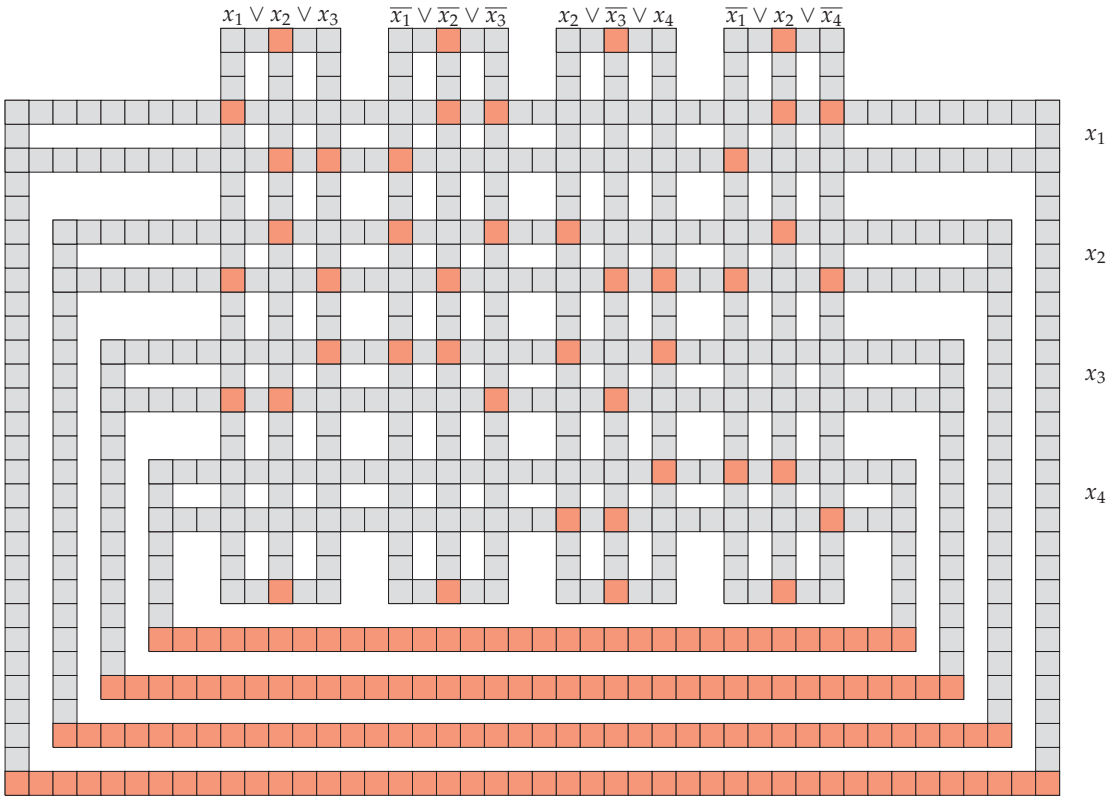


Figure 4.16: Example for the NP-hardness reduction using the *One-In-Three-3SAT* formula $(x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_2 \vee \overline{x_3} \vee x_4) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_4})$

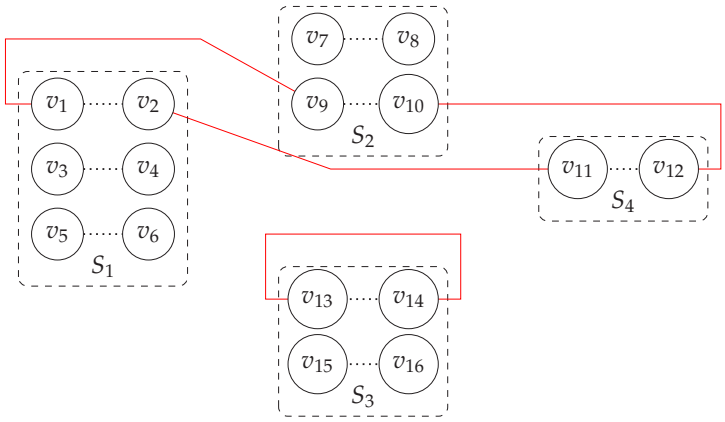


Figure 4.17: Example for Problem 7. The edges and edge weights are not drawn due to their huge amount but a possible matching solution is drawn in red. The pairs are displayed via dotted edges, e.g. $\{v_1, v_2\}$. Every cluster $S_1, \ldots, S_4$ has one matched pair.

lems. It is recommended to read Sec. 4.2.1 before this section in order to recognize the applicability.

The problem is similar to minimum weight perfect matching but there is also a pairing of vertices that enforces pairs to have the same number of edges in the matching. Further, similar to generalized TSP, not all vertices have to be matched but only at least one from a set of pairs (there are multiple of these sets). There is also a constraint on the edge weights that is related (but not equal) to the triangle inequality. This rather abstract problem can be stated formally as follows:

Problem 7. *Given*

- *a weighted graph* $G = (V, E)$ *with a cost function* $c : E \to \mathbb{R}^+$,
- *a partition of* V *into pairs* $P = \{p_1, \ldots, p_{|V|/2}\} \subset V \times V$ *with* $p_1 \dot\cup \ldots \dot\cup p_{|V|/2} = V$, *and*
- *a partition of* P *into sets* $S = S_1, \ldots, S_m$ *with* $S_1 \dot\cup \ldots \dot\cup S_m = P$.

Further, we assume

$$\begin{aligned} &\forall \{u, v\}, \{v', w\} \in E : \\ &\qquad \{v, v'\} \in P \Rightarrow \left(\{u, w\} \in E \wedge c(\{u, w\}) \leq c(\{u, v\}) + c(\{v', w\})\right) \end{aligned} \tag{4.1}$$

Wanted is a matching $M \subseteq E$ *of minimum weight* $\left(\sum_{e \in M} c(e)\right)$ *that fulfills*

- $\forall p_i \in P = \{u, v\} : |M(u)| = |M(v)| \leq 1$ *and*
- $\forall S_i \in S : \exists p_j = \{u, v\} \in S_i : |M(u)| = |M(v)| = 1$.

where $M(v) = \{e \in M \mid v \in e\}$.

An example is drawn in Fig. 4.17.

Theorem 9. *Problem 7 is NP-hard.*

Proof. We can express the cycle cover (Problem 1), which we proved to be NP-hard in the previous section, as an instance of Problem 7. For this we use the graph theoretic expression of the problem as described in Sec. 4.2.1 and for each field v we have the pairs $\{v_{north}, v_{south}\}$ and $\{v_{east}, v_{west}\}$. Hence, $S_v = \{\{v_{north}, v_{south}\}, \{v_{east}, v_{west}\}\}$. □

We can express Problem 7 by the following integer program:

$$\begin{aligned}
\min \quad & \sum_{e \in E} c(e) * x_e && && (4.2)\\
\text{s.t.} \quad & \sum_{e \in E(u)} x_e = \sum_{e' \in E(v)} x_{e'} = x_p && \forall p = \{u, v\} \in P && (4.3)\\
& \sum_{p \in S_i} x_p \geq 1 && \forall S_i \in S && (4.4)\\
& x_e \in \{0, 1\} && \forall e \in E && (4.5)\\
& x_p \in \{0, 1\} && \forall p \in P && (4.6)
\end{aligned}$$

The solution consists of all edges for which the corresponding variable $x_e, e \in E$ is 1. Eq. 4.3 states that if and only if for a pair $p = \{u, v\}$ the variable is set to 1, the two vertices u and v both have an edge in the matching (and also that both have the same amount of edges in the matching). Eq. 4.4 states that at least one pair of every set has to have edges in the matching.

Of course, we could have stated the integer program without the auxiliary variables $x_p, p \in P$ that indicate if a pair p has edges in the matching, but it is useful for the approximation algorithm. It does not only allow a simpler formulation but it also makes proving the approximation factor simpler. In a practical implementation, however, one can substitute these auxiliary variables.

The Match-Max-Pairs-Algorithm (Algorithm 1) solves the LP-relaxation of the integer program described in Eq. 4.2-4.6 and then does a matching on those pairs that have the highest values of their set. In the LP-relaxation, Eq. 4.5 and Eq. 4.2 are simply replaced by

$$\begin{aligned}
0 \leq x_e \leq 1 \quad & \forall e \in E && (4.7)\\
0 \leq x_p \leq 1 \quad & \forall p \in P && (4.8)
\end{aligned}$$

Algorithm 1: The Match-Max-Pairs-Algorithm

Data: $G = (V, E), c : E \to \mathbb{R}^+, P, S$

Result: An edge set fulfilling the constraints if LP is feasible

```
Solve LP 4.2-4.4, 4.7, 4.8;
V' ← ∅;
for S_i ∈ S do
    q_i ← maxarg_p{x_p | p ∈ S_i};
    V' ← V' ∪ q_i;
M ← optimal minimum weight perfect matching for G' = (V', {e ∈ E | e ⊂ V'});
return M;
```

Theorem 10. *Assuming all edge weights being expressible by a constant amount of bits, Algorithm 1 has a polynomial time complexity of $O(|V|^7)$ or $O(|E|^{3.5})$.*

Proof. The linear program has $O(|E|)$ $(\in O(|V|^2))$ variables and constraints and can be solved in polynomial time. Karmarkar's algorithm [69] is able to solve a linear program with n variables and numbers with $O(L)$ digits with $O(n^{3.5} * L)$ arithmetic operations (we assume L to be constant). This results in $(O(|E|^{3.5})$ (or $O(|V|^7))$ to solve our LP. The selection of the pairs with maximal value and the creation of the graph G' can be done in $O(|E|)$. The minimum weight perfect matching on G' can be calculated, e.g., with the famous blossom algorithm of Edmonds [70][71] in $O(|V|^2 * |E|)$. There exist algorithms with better complexity bounds, see the article of Cook and Rohe [72], but as the complexity of the linear program dominates, this does not matter for the theoretical consideration. □

Theorem 11. *The solution produced by Algorithm 1 is within* $2 * \max\{|S_i| \mid S_i \in S\}$ *times the optimum.*

Proof. We proof this theorem by showing that we can obtain a matching for $G' = (V', \{e \in E \mid e \subset V'\})$ (by rounding and repairing the LP-relaxation) which has a cost of at most $2 * \max\{|S_i| \mid S_i \in S\}$ times the objective value of the LP-relaxation.

For the following approach to work, we need to add loop-edges. Every vertex $v \in V$ gets a further edge $\{v, v\}$ with the weight $c(\{v, v\}) = \min\{c(\{v, u\}) + c(\{v, w\}) \mid \{u, w\} \in P \wedge \{v, u\}, \{v, w\} \in E\}$, i.e., an implicit generalization of Eq. 4.1 to loop-edges. In case that no such pair $\{u, w\}$ exists, the loop-edge is not necessary because they wouldn't be used in the next step because in this case, the skipping of pairs cannot create a loop. Such a loop-edge counts twice in Eq. 4.4 and Eq. 4.5, i.e., can only be selected with 0 or 0.5 where 0.5 suffices to cover v. These loop-edges are only needed for the intermediate solution and can later be replaced by regular edges without an increase of costs. It is easy to see, that the corresponding LP-relaxation is still a valid lower bound. The loop-edges are added after we solved the initial LP-relaxation to select the pairs. We call the new graph $G'_l(V', E'_l)$.

Due to the constraint in Eq. 4.4, it holds for every S_i that $x_{q_i} \geq 1/|S_i| \geq 1/\max\{|S_i| \mid S_i \in S\}$ in any feasible solution of the LP-relaxation. q_i is one of the pairs in S_i with maximal x_p value in the solution as defined in the algorithm. There might also be further $p \in P \setminus \{q_i \mid S_i \in S\}$ with $x_p > 0$. We can set these to zero without increasing the objective value and without changing any x_{q_i}. This is due to the constraint in Eq. 4.1 on the problem instance. The resulting solution only violates Eq. 4.4 by relaxing it to

$$\sum_{p \in S_i} x_p \geq 1/|S_i \quad \forall S_i \in S$$

but we fix this later by a simple multiplication.

The procedure in Algorithm 2 gets a value $\phi \geq 0$ and a pair $p \in P$ with $x_p \geq \phi$ and modifies the solution such that $x_p = \phi$, all variables are still within their range, no other pair variable has been modified, Eq.4.3 is still fulfilled, and the cost of the solution is not increased. As long as x_p is not zero, at least one of the three cases has to be valid:

- The pair uses (> 0) on both sides loop-edges
- The pair uses (> 0) a loop-edge on one side and a regular edge on the other

- The pair uses (> 0) regular edges on both sides

In the first case, we can directly reduce x_p by reducing the usage of both loop-edges. In the second case, we can simply move the loop which does not increase the costs because the loop can be seen as a path over a further pair and then we can skip the intermediate pair $p = \{u, v\}$ due to Eq. 4.1, i.e., $c(\{w, w\}) \leq c(\{w, s\}) + c(\{w, s'\}) \leq c(\{w, v\}) + c(\{u, u\}) + c(\{v, w\}) = c(\{w, v\}) + c(\{u, s\}) + c(\{s', u\}) + c(\{v, w\})$ for some $\{s, s'\} \in P$ (remember that a loop-edge counts twice). The third case is a direct application of Eq. 4.1. In the local modifications, it can be seen that Eq. 4.3 is not violated by the operations. Because we set a lower bound of $x_p - \phi$ and all the time $x_p \geq \phi$, x_p is never lowered below ϕ. Every case needs only to be checked once since the edge usages of incident edges is never increased during the procedure.

Algorithm 2: Procedure to lower a pair variable x_p to a specific value ϕ

Data: Pair $p = \{u, v\} \in P$, wanted value $\phi \geq 0$ for x_p with $\phi \leq x_p$

Procedure *LowerValue*(p, ϕ)**:**

`/* Remove smallest loop-edge (if existent) */`

$\gamma \leftarrow \min\{x_p - \phi, 2 * x_{\{u,u\}}, 2 * x_{\{v,v\}}\}$;

$x_p \leftarrow x_p - \gamma$;

$x_{\{u,u\}} \leftarrow x_{\{u,u\}} - 0.5 * \gamma$;

$x_{\{v,v\}} \leftarrow x_{\{v,v\}} - 0.5 * \gamma$;

`/* Replace possible remaining loop-edge */`

for $x \in \{u, v\}$ **do**

  for $e = \{w, x\}, e' = \{x, x\} \in E'_l$ **do**

    $e_p \leftarrow \{w, w\} \in E'_l$;

    $\gamma \leftarrow \min\{x_p - \phi, x_e, x_{e'}, 2 * x_{e''}\}$;

    $x_{e_p} \leftarrow x_{e_p} + 0.5 * \gamma$;

    $x_e \leftarrow x_e - \gamma$;

    $x_{e'} \leftarrow x_{e'} - 0.5 * \gamma$;

    $x_p \leftarrow x_p - \gamma$;

`/* Replace remaining edges */`

for $e = \{w, u\}, e' = \{v, w'\} \in E'_l$ **do**

  $e_p \leftarrow \{w, w'\} \in E'_l$;

  $\gamma \leftarrow \min\{x_p - \phi, x_e, x_{e'}\}$;

  $x_{e_p} \leftarrow x_{e_p} + \gamma$;

  $x_e \leftarrow x_e - \gamma$;

  $x_{e'} \leftarrow x_{e'} - \gamma$;

  $x_p \leftarrow x_p - \gamma$;

`/*` $x_p = \phi$ `*/`

Applying LowerValue$(p, 0)$ for all $p \in P \setminus \{q_i \mid S_i \in S\}$ results in $x_{q_i} \geq 1/|S_i|$ for all S_i and $x_p = 0$ for all $p \in P \setminus \{q_i \mid S_i \in S\}$. The corresponding solution is still a lower

bound on the optimal solution of the integer program.

We can multiply the solution by $\max\{|S_i| \mid S_i \in S\}$ resulting in $x_{p'_i} \geq 1$ for all S_i. The new solution has a cost of at most ω times the optimal solution and Eq. 4.5 is fulfilled again (but possibly with the usage of loop-edges). All variables x_p are now boolean and the only possibly unmet constraint is Eq. 4.5, i.e., the integrality of the edge variables.

If we now fix the values for $x_{p'}$ and remove all vertices (and their incident edges) that are not in V', we have a minimum perfect matching polytope for G'_l with loop-edges as stated below.

$$\begin{aligned} \min \quad & \sum_{e \in E'_l} c(e) * x_e && && (4.9)\\ \text{s.t.} \quad & \sum_{e=\{v,w\} \in E'_l} x_e + 2 * x_{\{v,v\}} = 1 && \forall v \in V' && (4.10)\\ & 0 \leq x_e \leq 1 && \forall e \in E'_l && (4.11) \end{aligned}$$

Because the previous solution is still feasible in this formulation, the optimal solution of this formulation is also at most ω the costs of the optimal solution of the problem.

We can show that the new polytope contains an half-integral optimal solution. The proof works analogously to the proof of the half-integrality of the classical matching polytope (see, e.g., Theorem 6.13 in the book of Cook et al. [73]). We create a bipartite graph $G'_d(V' \cup V'_d, E'_d)$ with for every $v \in V'$ there is a second vertex $v' \in V'_d$ and for every edge $e = \{v, w\} \in E'_l$ there are the edges $\{v, w'\}$ and $\{v', w\}$ in E'_d, see Fig. 4.18. A loop-edge

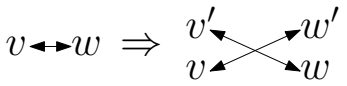


Figure 4.18: Creating bipartite graph

$\{v, v\} \in E'_l$ is simply replaced by a single edge $\{v, v'\} \in E'_d$. Because there are no edges within V' or V'_d, the graph is bipartite. For bipartite graphs, the matching polytope is known to be integral. By assigning the edge in G'_d the same values as for the solution on G'_l (doubling for loop-edges), we obtain a feasible solution with twice the costs on G'_d. The optimal solution in G'_d, hence, at most twice as expensive as the optimal solution in G'_l. The optimal solution in G'_d (which is integral) can be transformed to a solution of half the costs for G'_l by assigning each edge in G'_l half the sum of the corresponding two edges (or one edge for loop-edges) in the solution for G'_d. This solution is half-integral and optimal. Hence, we have an optimal solution where regular edges are selected by 0, 0.5, or 1 and loop-edges are selected by 0 or 0.5.

If we double such a solution, we obtain an integral solution with at most $2 * \omega$ the costs of the original linear program. Every vertex has either two regular edges selected with 1 each, a single regular edge selected twice, or a loop-edge selected with 1. Every loop-edge can be replaced by the corresponding two edges used for its weight definition. We can remove redundant visitations by simply skipping the second visit without a cost increase due to Eq. 4.1. As long as there is a pair visited multiple times, we can find a cycle over pairs that visits a redundantly visited pair but visits itself every pair at most

once (if it visits a doubly visited pair twice, we could shorten the cycle). If all its pairs are visited redundantly, the cycle can simply be removed without leaving any pair unvisited. Otherwise, there exists a multiple visited pair succeeded by a uniquely visited pair for which we can skip the multiple visited pair without creating a cycle, and hence reduce the number of redundant visitations (see Fig. 4.19). This can be repeated until no redundantly visited pair is left. Finally, the pairs that have been added for replacing the loop-edges can be skipped without creating a new loop-edge since no vertex has more than one edge left.

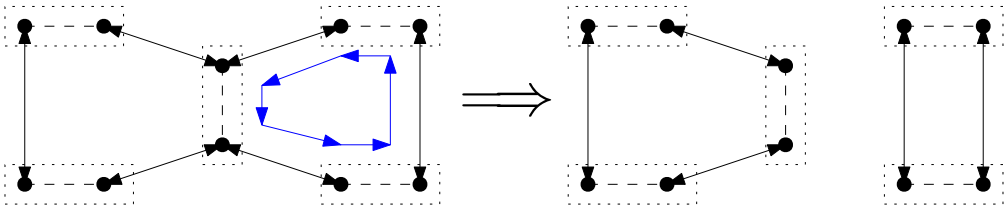


Figure 4.19: Redundant visitations can be skipped without using loop-edges by a shortest cycle (blue) that has at least one multiple visited pair. In case all pair a visited multiple times, the whole cycle can be removed.

In the end, we have a solution that is a matching for G' and has at most $2 * \max\{|S_i| \mid S_i \in S\}$ the cost of the objective value of the linear relaxation. As the linear relaxation provides a natural lower bound, this concludes the proof. □

This result leads to the following result:

Theorem 12. *Algorithm 1 provides a 4-approximation algorithm for cycle cover in 2-dimensional grid graphs (Problem 1)*

Proof. We have already seen in Theorem 9 that Problem 7 can be reduced to Problem 1. This reduction needs only two pairs per fields resulting in $\max\{|S_i| \mid S_i \in S\} = 2$. Algorithm 1 provides a 4-approximation as proven in Theorem 11. Implicitly added edges for Eq. 4.1 have to be replaced by the original edges in the end. □

In the next section, we see the approximation algorithm of Arkin et al. [3] which provides a better approximation ratio and a significant better runtime for 2-dimensional grid graphs. However, our technique that uses Algorithm 1 can also be used for the other problem variants while the technique of Arkin et al. [3] cannot.

4.3.2 Full coverage

In this section, we consider approximation algorithms for full coverage cycle covers and tours. We already showed quickly (Theorem 12) that the Match-Max-Pairs-Algorithm (Algorithm 1) can be used to obtain a 4-approximation for the full coverage cycle cover problem in 2-dimensional grid graphs (easily generalizable, e.g., to a 6-approximation in hexagonal grid graphs). In 2-dimensional grid graphs, the minimum strip cover technique of Arkin et al. [3] is superior such that we first review their technique. Afterward, we consider

the generalization for more general grid graphs and show that our new method provides for this case better approximation factors. The main problem is to obtain a cycle cover; a tour can be easily obtained by a simple cycle merging technique. In Sec. 4.3.3, we discuss why their technique fails for subset (and penalty) coverage. For geometric instead of grid instances, their technique fails simply because strips are a concept of grid graphs.

2-dimensional grid graphs

Arkin et al. [3] show an approximation algorithm for cycle cover that achieves an approximation factor of 2.5 on the number of turns and 4 on the length (bicriteria). They also show how to connect these cycles in order to obtain a tour, resulting in an approximation factor of $3.75 (= 1.5 * 2.5)$ on the number of turns and 4 on the length (bicriteria).

Theorem 13 (Arkin et al. 2005 [3])**.** *There is an approximation algorithm for cycle cover in 2-dimensional grid graphs that provides a solution with at most* 2.5 *times the turn costs and* 4 *times the length costs of an optimal solution with a runtime in* $O(|V|^{2.376} + b^3)$ *where* b *is the number of boundary segments.*

Proof. For simplicity, we neglect the exact runtime and refer for the details to the original paper. The idea is to first calculate an optimal strip cover (which is a lower bound on the number of turns) and then connected the end of the strips via a minimum weight perfect matching. It can easily be seen that an optimal strip cover is a lower bound for the number of turns in a cycle cover since we can obtain a strip cover with t strips from a cycle cover with t turns: simply use two successive turns as strip direction. The optimal strip cover can also efficiently calculated in 2-dimensional grid graphs: consider every possible (full) strip as a vertex and connect two strips if they intersect. An optimal vertex cover provides the optimal strip cover and can be calculated in polynomial time since the corresponding graph is bipartite. Now we connect the ends of the strips (which are wall fields) via a minimum weight perfect matching. The weight of the edges are the respective connection costs. Because wall fields can only be crossed straight in one direction, we can calculate an optimal cycle cover that covers all end fields via a 2-factor (see Sec. 4.2.1). The minimum weight perfect matching (excluding the turn at the end fields) has at most half the costs and thus half the costs of an optimal full cycle cover. For every strip we have to make at most two additional turns (one at each end) resulting in at 2.5 times the turns of the optimum.

The bound on the length can be achieved by Theorem 15. □

Theorem 14 (Arkin et al. 2005 [3])**.** *Every cycle cover can be transformed into a tour of at most* 1.5 *times the turn costs.*

Proof. We can merge two adjacent cycles (two covered fields of them are either identical or adjacent) with only two additional turns: In case the two cycles are intersecting, the solution is obvious. In case the two cycles do not intersect, it can be argued that in one adjacent field pair there has to be a turn. Such a pair can again easily be connected with only two additional turns. As every cycle already has four turns itself, the resulting tour has at most 1.5 times the number of turns. □

Observation 2. *There are optimal cycle covers whose optimal tour has* 1.5 *times the number of turns.*

Proof. In Fig. 4.20 we see a construction that can come arbitrarily close to 1.5. While the cycle cover can get along with the corner fields, the tour needs two additional turns for each vertical block in the intersections of the vertical blocks with the horizontal blocks.

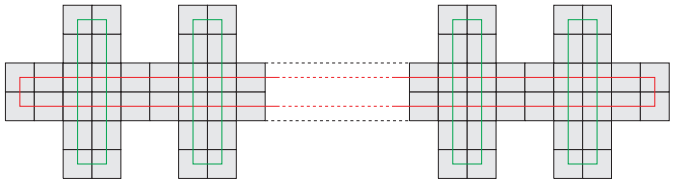


Figure 4.20: The optimal tour has two additional turns per green cycle in order to connect the vertical blocks with the horizontal block.

□

Observation 3. *There are optimal cycle covers that lead to tours with 1.5 times the number of turns when connected using Theorem 14 while the optimal tour has the same number of turns as the cycle cover.*

Proof. The construction in Fig. 4.21 can come arbitrarily close to 1.5. □

Figure 4.21: The optimal tour has the same amount of turns as the optimal cycle cover (red) but applying Theorem 14 results in two additional turns for each of the flat cycles.

Theorem 15 (Arkin et al. 2005 [3])**.** *Every tour can be transformed such that every vertex is visited at most* 4 *times and the turn costs are not increased.*

Proof. In case a field is visited more than four times a simple local modification can be applied. See the original paper for an illustration and details. □

Theorem 16 (Arkin et al. 2005 [3])**.** *There is an approximation algorithm for full coverage tours in 2-dimensional grid graphs that provides a solution with at most* 3.75 *times the turn costs and* 4 *times the length costs.*

Proof. Combine Theorem 13, 14, and 15. □

Other grids

We now consider other grids, in especially 3-dimensional and hexagonal grid graphs. A minimum strip cover is still a lower bound on the number of turns in a cycle cover but the method for calculating it in 2-dimensional grid graphs does not work here. If every field in the grid lies on at most ω full strips (i.e., 3 for 3-dimensional grid graphs and

hexagonal grids), a simple ω-approximation can be obtained for it: We greedily place a rook on a field that is not yet seen by any rook until all fields are covered. The number of rooks is a lower bound on the number of necessary strips. We can obtain a strip cover by using all ω strips of each rook. This provides a fast and simple $4 * \omega$ approximation algorithm for the cycle cover problem by simply using every strip as an individual cycle. For 2-dimensional grid graphs, we were able to connect the strips cheaply due to the fact that we can calculate an optimal boundary cover. For graphs with more distinct strips, this, unfortunately, does not work anymore. However, our Match-Max-Pairs-Algorithm can still be applied and needs only one pair for every possible full strip/orientation.

Theorem 17. *In a grid graph where every field lies on at most ω strips, there exists a $2 * \omega$-approximation algorithm for full cycle covers.*

Proof. This approximation factor can be achieved by our Match-Max-Pairs-Algorithm (Algorithm 1). Each strip results in two opposite directions a field can be entered or left. By using a representation as in Sec. 4.2.1, we can create an instance for the Match-Max-Pairs-Algorithm where the vertices of every two opposite directions are a pair. With at most ω pairs per field, this leads to a $2 * \omega$ approximation algorithm by Theorem 11. □

For 3-dimensional grid graphs and hexagonal grid graphs, we can deduce the following result:

Corollary 1. *There is a 6-approximation for full cycle covers in 3-dimensional grid graphs and in hexagonal grid graphs.*

In 3-dimensional grid graphs, there doesn't need to be two adjacent fields with a turn between two adjacent cycles. This result in higher costs for connecting a cycle cover to a tour compared to 2-dimensional grid graphs.

Theorem 18. *A full cycle cover in a 3-dimensional grid graph can be connected to a tour with at most twice the cost.*

Proof. There have to be at least two adjacent cycles if the cycle cover is not a tour. We can connect these two cycles with four turns (and two additional transitions). Because the cycle cover afterward has one cycle less and a cycle itself has at least four turns and a length of two, we can charge it to the 'removed' cycle. Since every cycle is only charged once, the resulting tour has at most twice the cost. □

Observation 4. *There are 3-dimensional grid graphs whose optimal full tours have twice as many turns as the optimal full cycle cover.*

Proof. A corresponding grid graph can be built as follows: For i starting at zero and arbitrarily incremented by 1, draw the fields $(1, 0, 2 * i)$, $(0, 0, 2 * i)$, $(-1, 0, 2 * i)$, $(1, 0, 2 * i + 1)$, $(0, 0, 2 * i + 1)$, $(-1, 0, 2 * i + 1)$ if i is even and $(0, 1, 2 * i)$, $(0, 0, 2 * i)$, $(0, -1, 2 * i)$, $(0, 1, 2 * i + 1)$, $(0, 0, 2 * i + 1)$, $(0, -1, 2 * i + 1)$ if i is odd. □

It can be seen that this connection-method works on general grid graphs.

Theorem 19. *Given a full cycle cover in an arbitrary (n-dimensional, hexagonal, etc.) grid graph, then we can connect the full cycle cover to a full tour of at most twice the costs.*

Proof. If there are more than two cycles in the cycle cover, there are at least two adjacent cycles, i.e., there exists an edge between two vertices of them. We can put a cheapest possible cycle on this edge that has a u-turn on both cycles. If a cycle lies on another cycle with a u-turn, the cycles can be merged without additional costs. We proof this for the geometric version in Lemma 8, which also contains the discrete version for grid graphs. For m cycles, at most $m - 1$ such cheapest possible cycles have to be created, hence the added costs are lower than the original costs. □

Corollary 2. *In a grid graph where every field lies on at most ω full strips, there exists a $4 * \omega$-approximation algorithm for full tours.*

4.3.3 Subset coverage

In this section, we consider approximation algorithms for the subset cycle cover and tour problems. We first show the problems of the strip cover method with this problem variant and then show that the Match-Max-Pairs-Algorithm can directly be applied resulting in a constant factor approximation algorithm for the subset cycle cover. Because cycles no longer need to be adjacent, we need a new cycle connecting technique. For this, we can use a doubled minimum spanning tree on the cycles resulting also in a constant factor approximation algorithm for subset tours.

We can easily generalize the strip cover to the subset variant. Instead, for a minimum amount of strips that cover all fields we now only search for a minimum amount of strips that cover at least a specific subset of fields. For 2-dimensional grid graphs we can even obtain an optimal such subset strip cover using the same 'vertex cover in bipartite graph'-technique but leaving out all edges for fields not in the subset. Obviously, the subset strip cover is also a lower bound on the number of turns in subset cycle cover. However, not only do we get a problem with the length of the tour but also the cost argument of connecting the end fields of the strips does no longer work: A long strip might cover only a single field of the subset and arguing that there is a technique to reduce the length of the tour if a field is visited too often does also not result in an approximation factor as a field might not have to be visited at all. Even if we concentrate only on the number of turns, the 2.5 approximation factor cannot (at least not directly) be achieved, see Fig. 4.22. The argument was based on the fact that the end fields have to be covered anyways and that they are at a wall that limits the possibilities. Now the end fields of the strips possibly no longer have to be covered and if we crop the full strips, the end fields do not longer have to be at a wall. A doubling of the strips at least provides a simple and fast 4-approximation algorithm on the number of turns (but not on the length).

On the other hand, Algorithm 1 is still able to provide a 4-approximation algorithm on the number of turns and the length for subset cycle cover in 2-dimensional grid graphs. The only difference to its application on cycle cover for full coverage is that we do not add those vertices and pairs corresponding to fields not in the subset. Only a new technique

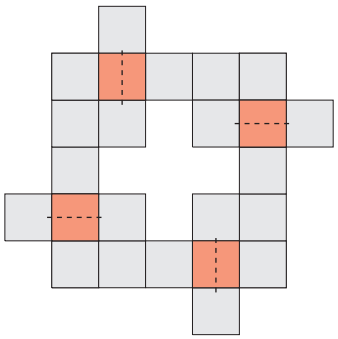


Figure 4.22: An example where a matching on an optimal subset strip cover (dashed lines) only provides a $3 * \text{OPT}$ solution on the number of turns for the subset cycle cover. If the strips are not cropped, it is even $4 * \text{OPT}$.

for connecting cycles to a tour is needed since there no longer have to be two adjacent cycles.

Theorem 20. *There is a 4-approximation algorithm for subset cycle cover in 2-dimensional grid graphs with a runtime in $O(|V|^7)$.*

Proof. We already mentioned in Sec. 4.2.1 that we can simply leave out those fields that do not have to be covered. These fields are now only implicitly used by edges whose corresponding path goes through them. This means, that the argument in Theorem 12 can still be used. Calculating the shortest paths between all four directions in all fields and creating the instance can be done in negligible time compared to the runtime of $O(n^7)$ (with $n = 4 * |V|$) for the Match-Max-Pairs-Algorithm. □

Note that the runtime of the expensive part, the linear program in the Match-Max-Pairs-Algorithm, is only dependent on the size of the subset. On the full set of fields, only an all-path-shortest-path algorithm has to be executed to obtain the representation for the Match-Max-Pairs-Algorithm.

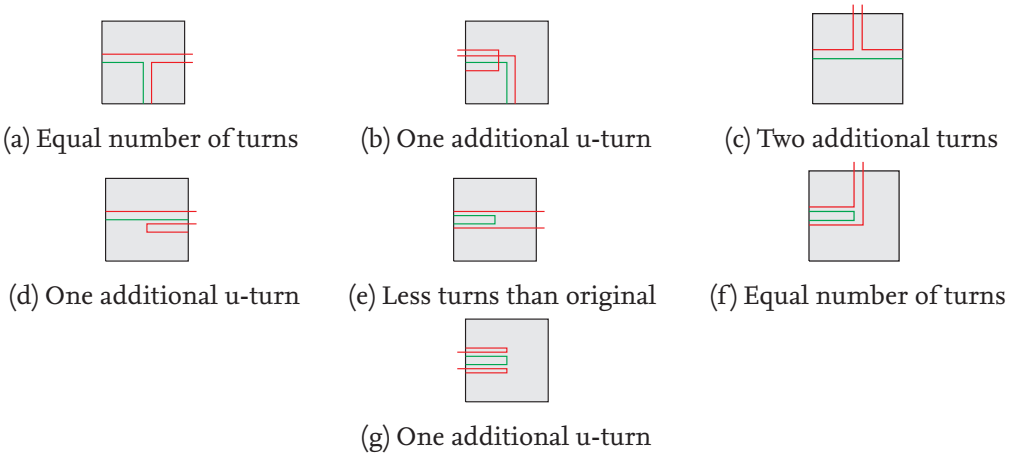
The result can easily be generalized to other grids:

Theorem 21. *There is a $2 * \omega$-approximation algorithm with a runtime in $O((\omega * |V|)^7)$ for subset cycle cover where every field has maximal ω orientations.*

Proof. As for 2-dimensional grid graphs but now every field in the subset might have ω pairs. □

The argument for connecting cycles used for full coverage no longer holds because cycles do not have to be adjacent in this problem. Indeed, they can be far apart. By considering each cycle as a vertex, we can connect them by a doubled minimum spanning tree.

Theorem 22. *A subset cycle cover of cost c can be transformed into a subset tour with costs at most $2 * c + 2 * OPT$ where OPT refers to the costs of the optimal tour.*



(a) Equal number of turns (b) One additional u-turn (c) Two additional turns

(d) One additional u-turn (e) Less turns than original (f) Equal number of turns

(g) One additional u-turn

Figure 4.23: Connecting a double edge of the minimum spanning tree to a cycle. The part of the cycle in green and the new connection from the minimum spanning tree in red.

Proof. Consider a minimum spanning tree in a complete graph where every cycle in the cycle cover is a vertex. The weights of the edges correspond to the costs of connecting the two cycles by a simple path, neglecting the turn costs of entering and leaving the cycle. If two cycles share the same field, the cost of the connecting edge is zero. Obviously, the cost of this minimum spanning tree is a lower bound for the optimum. Doubling the minimum spanning tree provides a tour but besides the cost of the cycle cover and twice the minimum spanning tree, there are also turn costs for connection. These turn costs are at most 4 per edge in the minimum spanning tree (simple distinction of cases as can be seen in Fig. 4.23). Since there are less such edges than cycles and every cycle has already at least turn costs of 4, the proof is concluded for 2-dimensional grid graphs. For other grid graphs this holds too but, e.g., for hexagonal grid graphs, a u-turn has a cost of 3 but a cycle also has to have at least 6 turns leading to the same result. □

Note that this is a very rough upper bound as can be seen in the proof. We only double the costs of the cycle cover to pay for the connection cost of maximally 4 turns per edge of the minimum spanning tree and there are fewer edges than cycles in the cycle cover. These costs could actually also be integrated into the edge weights of the minimum spanning tree graph to minimize these additional costs in practice but it is difficult to argue an improvement of the theoretical bound. In our experiments for the geometric problem variant (Sec. 5.5), the corresponding approximation algorithm that uses the same technique was even for the worst instance within 1.484 times the optimum.

Theorem 23. *There is a 10-approximation algorithm with a runtime in $O(|V|^7)$ for subset tours in 2-dimensional grid graphs.*

Proof. Apply Theorem 22 on Theorem 20. The connection of the cycle cover with the minimum spanning tree can obviously be done in time much less than $O(|V|^7)$. □

This can easily be generalized to

Theorem 24. *For a grid graph where every field has at most ω different covering full strips, there is a $4 * \omega + 2$-approximation algorithm with a runtime in $O((\omega * |V|)^7)$ for subset tours.*

Proof. Trivial adaption of Theorem 23. □

4.3.4 Penalty coverage

In this section, we consider approximation algorithms for penalty coverage. We give constant factor approximations for penalty cycle covers and penalty tours using the Match-Max-Pairs-Algorithm of Sec. 4.3.1 and a 2-approximation algorithm for the prize-collecting Steiner tree by Goemans and Williamson [53].

In penalty coverage, no field has to be covered at all but there is an individual penalty for each uncovered field. An adaption of the strip cover technique now becomes even harder than for subset coverage. The Match-Max-Pairs technique, however, can easily be adapted to penalty cycle cover by simply adding an additional pair per field that has a single edge with the cost of the penalty. This results in constant factor approximations for penalty cycle covers. For connecting these cycles, we cannot use the doubled minimum spanning tree technique as for subset coverage: For example, if the penalty cycle cover consists of two distant cycles, it might be cheaper not to connect them but select the better cycle as tour and discard the other cycle. Doubling a prize-collecting Steiner tree instead of a minimum spanning tree, however, allows us to also provide a constant factor approximation for penalty tours.

Paying the penalty for an uncovered field can also be seen as covering this field by an artificial cycle (that covers exactly this field) with the cost of the penalty. The adaption of the algorithm for penalty cycle cover, hence, is rather simple.

Theorem 25. *There is a 6-approximation algorithm with a runtime in $O(|V|^7)$ for penalty cycle cover in 2-dimensional grid graphs.*

Proof. As for the full coverage and subset coverage version (Theorem 12 and 21), we have for every field v the pairs $\{v_{\text{north}}, v_{\text{south}}\}$ and $\{v_{\text{east}}, v_{\text{west}}\}$ with edge costs as described in Sec. 4.2.1. Additionally, we add for every field the pair $p_{\text{penalty}} = \{v_{p1}, v_{p2}\}$ and the edge $\{v_{p1}, v_{p2}\}$ with cost $\rho(v)$ where $\rho(v)$ defines the penalty if v is not covered. Because it is the only edge of this pair, it does not interfere with the edge weight constraints. In the case that Algorithm 1 selects p_{penalty}, the field is not covered and the penalty is paid. For all other fields we obtain a normal cycle cover. □

A simple observation is that fields with a penalty larger than the cost of a minimum cycle are always kept because simple 2-field-cycles only for these fields are cheaper than paying the penalties. It might be possible that this observation can be used to get rid of the additional penalty pair, and thus improve the approximation factor. Of course, we can also generalize the approximation algorithm to other grids.

Theorem 26. *If every field is covered by at most ω strips, there is a $2 * (\omega + 1)$ approximation algorithm with a runtime in $O((\omega * |V|)^7)$ for penalty cycle cover.*

Proof. Straight forward generalization of Theorem 25. □

For obtaining a tour, we now need to select a set of cycles of the penalty cycle cover and connect them.

Theorem 27. *There is a 16-approximation algorithm with a runtime in $O(|V|^7)$ for penalty tours in 2-dimensional grid graphs.*

Proof. This factor consists of 6 for the penalty cycle cover, $2 * 2$ for twice the approximated prize-collecting Steiner tree, and again 6 for connecting the cycles to the doubled tree which costs less than the cycle cover. We now look onto the details.

We start by calculating a penalty cycle cover approximation using Theorem 25. All fields for which the penalty is paid in this cycle cover are dismissed and the remaining set of fields is denoted by V'. We now calculate a 2-approximation of a prize-collecting Steiner tree by the algorithm of Goemans and Williamson [53] with a time complexity of $O(n^2 \log n)$ on the fields in V' where the penalties are unchanged but we have a different set of edges. There is an edge between any pair of fields that are in different cycles with the cost of the shortest connecting path neglecting the orientations at the ends. Between any two fields of the same cycle, there is an edge without any cost. Obviously, the cost of the resulting tree is a lower bound on the optimal tour. We dismiss all cycles for which there is no field in the tree because it already has paid the penalty of all its fields. From the tree, we can select edges that connect all remaining cycles by simple paths. If two cycles are going through the same field, we assume a virtual path of zero length between them. Doubling these, we obtain a tour with the cost of the penalty cycle cover ($\leq 6 * \mathrm{OPT}$) plus twice the cost of the prize-collecting Steiner tree ($\leq 2 * 2 * \mathrm{OPT}$) plus the turn costs for connecting the cycles via the edges of the tree. These are at most the cost of the penalty cycle cover (along the same line of reasoning as in Theorem 22). This results in an approximation factor of $2 * 6 + 2 * 2 = 16$. The runtime is again obviously dominated by the Match-Max-Pairs-Algorithm. □

Again, this can be easily generalized to other grids.

Theorem 28. *If every field has at most ω covering full strips, there is a $4 * (\omega + 1) + 4$-approximation algorithm with a runtime in $O((\omega * |V|)^7)$ for penalty tours.*

Proof. Straight forward adaption of Theorem 27. □

4.4 Integer programming

In this section, we consider how to obtain optimal solutions for the cycle cover and tour variants using integer programming. Modern solvers as CPLEX [8] have been successfully applied to many problems but their performance varies with the problem formulation and separation methods. The performance is often hard to predict and two different approaches for the same problem can have unexpected and strongly differing runtimes. For this reason, we use multiple formulations for the cycle cover problem and different subtour separation constraints for obtaining a tour. In Sec. 4.4.1, 4.4.2, and 4.4.3, we describe

the integer programming formulations for the different problem variants. The formulations for the different problem variants are similar such that we only show them in detail for the full coverage variant and then only discuss the necessary changes for subset coverage and penalty coverage. We focus only on 2-dimensional grid graphs, but the adaption to other grids is straightforward. The experimental evaluation can be found in Sec. 4.5.

4.4.1 Full coverage

We now state three different integer programming formulations for the full coverage cycle cover and tour problem. The only differences between the cycle cover and the tour formulation is that for the tour formulation we iteratively add additional constraints until the cycles in the cycle cover connect to an optimal tour. This equals the classical integer programming approach for the traveling salesman problem (see e.g. [24]).

de Assis and de Souza [4] have already proposed an IP-formulation for the tour problem which, however, only was able to solve small instances of around 70 points. It is similar to the third formulation for cycle cover but strongly differs in the subtour separation for obtaining a tour. As we are able to solve much larger instances with this new separation, it is likely that their separation method was the bottleneck. They did not provide any experiments for the cycle cover problem. Maurer [7] on the other hand provides a formulation for the cycle cover problem which was able to solve larger instances in a 50×50 grid graph to optimality. It equals the cycle cover version of the first formulation. He did not consider tours and did not provide sufficient experimental data. The second formulation is similar to the linear program used in Alg. 1. The subtour separation constraint of it is similar to a common subtour separation in the traveling salesman formulation.

There are many other possible formulations of the problem. However, the first formulation seemed to be the most promising in first experiments and has been successfully applied to the cycle cover problem by Maurer [7]. The second formulation has been chosen due to its similarity to classical combinatorial problems (Matching, TSP) and has has a simple subtour separation that has already been successfully applied to the traveling salesman problem. The third formulation has been included because it has a nice graph theoretical representation and to provide a connection to the previous work of de Assis and de Souza [4].

Formulation 1

A field can only be passed in a very limited amount of ways, e.g. straight horizontal, south-west-turn, or an u-turn to the north. In the first formulation, we provide a non-negative integer variable for each of these possibilities per field that represents how often a field is passed in this way. To ensure a full coverage, we demand that every field is passed at least once in one of these ways, i.e., at least one of the variables per field is greater than zero. Of course there are dependencies between two adjacent fields and their ways of passing. If a field is entered/left a specific amount of times from east, the field right to it has to be entered/left from west the same amount of times. These variables and constraints have been visualized in Fig. 4.24 and are sufficient to express the cycle cover problem as an

integer program.

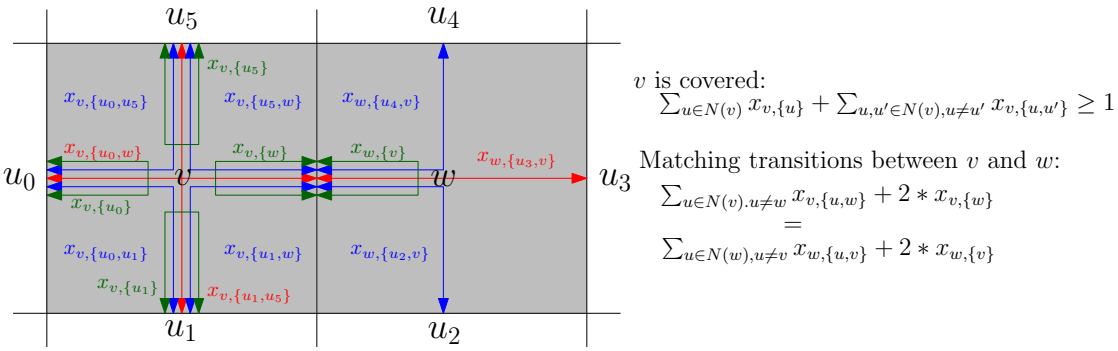


Figure 4.24: The IP-formulation 1 focused on the field v. The first constraint enforces v to be visited at least once. The second enforces equal passes between v and w.

We now state the formulation more formally, beginning with the definition of the variables and the cost function. Let $G = (V, E)$ be the grid graph of the problem instance. For every field $v \in V$ with the two neighbors $u, w \in N(v)$ there is a variable $x_{v,\{u,w\}} \in \mathbb{N}_0$. This variable stands for passing v by entering from u and leaving through w (or entering from w and leaving through w as this is undirected). For example, the variable for passing the field $v \in V$ via an u-turn from $w \in N(v)$ is denoted by $x_{v,\{w,w\}} = x_{v,\{w\}}$. The cost function c is defined as follows

$$\forall x_{v,\{u,w\}} : c(x_{u,\{u,w\}}) \quad = \quad \begin{cases} 2+d & u = w \\ 0+d & u.x = w.x \dot{\vee} u.y = w.y \\ 1+d & \text{otherwise} \end{cases} \tag{4.12}$$

where, e.g., $u.x$ denotes the x coordinate of u and $d \in \mathbb{R}_0^+$ is the distance cost for a field transition. As the scaling of the costs does not matter, this is actually a linear combination of distance and turn costs. Using the κ and τ weights, d would be κ / τ and the final solution would be multiplied by τ.

The integer program for the cycle cover problem is stated as follows. The additional constraints for the tour problem are considered afterward.

$$\min \quad \sum_{v\in V} \sum_{\{u,w\}\subseteq N(v)} c(x_{v,\{u,w\}}) * x_{v,\{u,w\}} \tag{4.13}$$

$$\text{s.t.} \quad \sum_{\{u,w\}\subseteq N(v)} x_{v,\{u,w\}} \geq 1 \qquad \forall v \in V \tag{4.14}$$

$$2 * x_{v,\{w\}} + \sum_{u\in N(v),u\neq w} x_{v,\{w,u\}} = 2 * x_{w,\{v\}} + \sum_{v\in N(u),u\neq v} x_{w,\{w,v\}} \qquad \forall \{v,w\} \in E \tag{4.15}$$

$$x_{v,\{u,w\}} \in \mathbb{N}_0 \qquad \forall v \in V, \{u,w\} \subseteq N(v) \tag{4.16}$$

The constraint in Eq. 4.14 enforces each field to be visited at least once and the constraint in Eq.4.15 enforces that each side of a field is entered as much as left. Note that during this whole chapter $\{u, w\} \subseteq N(v)$ also includes the case $u = w$ and every set is, independent of the identifiers, only considered once in an iteration of the sum. One can actually also add the constraint that every field is only allowed to be passed at most four times due to the result of Arkin et al. [3] cited in Theorem 15.

A solution of the integer program does not necessarily yield a unique cycle cover. However, a cycle cover that uses exactly these passings (and hence has exactly these costs) can easily be obtained. For this, only an unused passing has to be selected and from this one, greedily the next unused incident passing until the cycle is closed. This is repeated until no unused passings remain. In case of a tour, these cycles can be connected for free at their intersection points, see Fig. 4.25. This is similar to Hierholzer's algorithm for eulerian cycles [74].

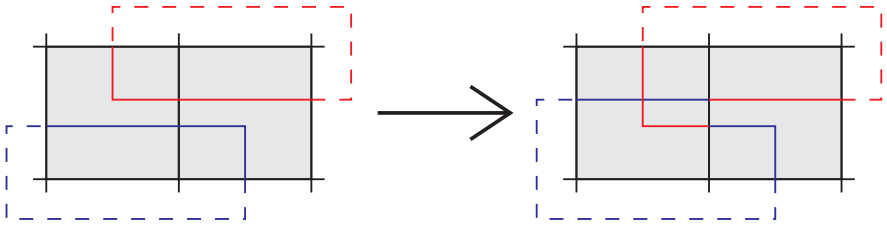


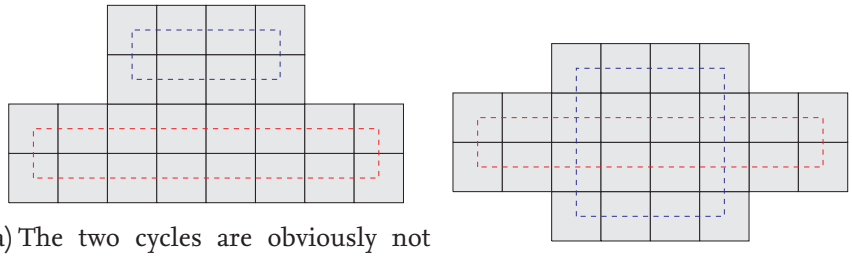
Figure 4.25: If the solution of the integer program is actually a tour, the cycles can be connected for free

In general, the integer program above only provides a cycle cover. If we want to obtain a tour, we have to add further constraints that exclude solutions that do not provide a tour. These constraints are selectively added iteratively as long as the integer program does not yield a tour. This could also be considered as iteratively forcing the cycles to connect. It is important to know how cycles can be 'disconnected' in order to prohibit such 'subtours'. While in the traveling salesman problem two subtours are 'disconnected' if they do not share any vertex, in our problem two cycles can be disjunct even if they both pass the same field. Fig. 4.26 presents these two ways of not being not connectable.

Eliminating subtours as in Fig. 4.26a can be simply done by demanding that there have to be two passings that leave the subtours. This equals a classical subtour elimination constraint for the traveling salesman problem and is considered next. Because this is not sufficient, we consider another separation technique afterward that can even deal with subtours as in Fig. 4.26b.

Simple insufficient subtour separation constraint We start with a simple separation constraint that is able to prohibit non-intersecting subtours of the kind as in Fig. 4.26a but not those of kind as in Fig. 4.26b.

Let C be a subtour that is not crossed by any other cycle. It is easy to see that there has to be a variable used that allows to leave the set of fields $V' = \{v \in C\}$ covered by C.



(a) The two cycles are obviously not connected as their fields are disjunct.

(b) These two cycles are also not connected even though they intersect.

Figure 4.26: Detecting not connectable cycles is not as simple as for the traveling salesman problem.

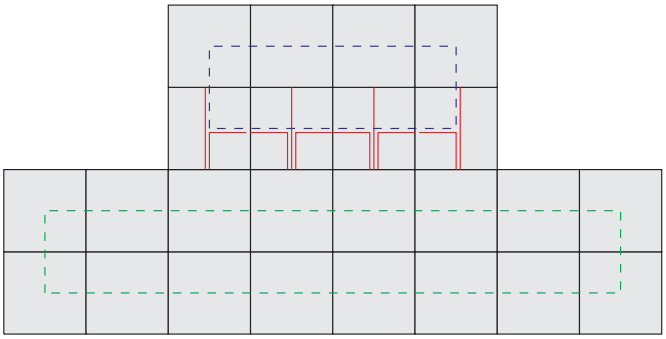


Figure 4.27: The first separation constraint on the cycle above. At least one of the red passings has to be used for the blue cycle.

$$\sum_{v \in V', \{w,w'\} \subseteq N(v), w \notin V'} x_{v,\{w,w'\}} \geq 1 \quad \forall V' \subsetneq V, V' \neq \emptyset \tag{4.17}$$

Actually this is valid for any real non-empty subset $V' \subsetneq V$ of fields independent of any cycle and hence C might also consist of, e.g., two intersecting cycles as long as C does not fully cover V.

It is not practical to add all possible constraints of form as in Eq. 4.17 in advance. Instead, we calculate after each iteration a subgraph $G' = (V, E' \subseteq E)$ with $E' = \{\{u,v\} \mid \{u,v\} \in E, \exists w \in V : x_{u,\{v,w\}} \geq 1 \vee x_{v,\{u,w\}} \geq 1\}$. If there is more than one connected component in G', apply the separation constraint for each of these. An example can be seen in Fig. 4.27. As already mentioned, this separation constraint is unfortunately not sufficient. We now consider another, this time sufficient, separation constraint.

Advanced sufficient subtour separation constraint We need a separation constraint that is also able to separate subtours as, e.g., in Fig. 4.26b. The following separation constraint is applicable to all subtours C that have a field f only covered by C and another field f' not covered by C. Such two fields do not exist for all cycles of a (not necessarily optimal) cycle cover but in every cycle cover that does not contain a tour there is at least one such cycle.

The separation constraint enforces that either f is passed differently or there is a turn in a field only passed straight by C or a currently unused variable is used in one of the other fields of C. Let F_s be the set of fields only passed straight by C excluding f. Other cycles might also pass fields in F_s but not by a simple turn as otherwise they would be connected with C. Let $T(v)$ be the simple turn variables for the field $v \in V$ and x' denote the value of a variable in the current solution.

$$\sum_{\{u,w\} \subseteq N(f), x'_{f,\{u,w\}}=0} x_{f,\{u,w\}} + \sum_{t \in T(v), v \in F_s} t + \sum_{v \in C \setminus (F_s \cup \{f\}), u \neq w \in V, x'_{v,\{u,w\}}=0} x_{v,\{u,w\}} \geq 1 \quad (4.18)$$

An example of this separation constraint can be seen in Fig. 4.28. While for the previous separation constraint, we could theoretically use any subset of fields, this separation constraint needs an actual solution to be useful. We now show the correctness and completeness of this separation constraint.

Lemma 6. *Let S be the optimal solution of the integer program and possibly a set of additional constraints that do not exclude an optimal tour. Further, S does not have a covering tour and has a cycle C that covers a field f exclusively and does not cover a field f'. Adding a separation constraint as in Eq. 4.18 separates S but does not exclude any optimal tour.*

Proof. It is obvious to see that the separation constraint separates the current solution because all variables in it are currently unused but the separation constraint enforces at least one of them to be used. We now proof that no optimal solution is excluded by contradiction: Let S^* be an excluded optimal solution. There is a path in it connecting f and f'. If it passes f differently than C, the constraint is already fulfilled as only C passes f in S. So the beginning of the path coincides with C. Let s be the first passing where they differ. If the corresponding field is in F_s, s must be a turn to leave the cycle which would automatically fulfill the constraint. If it is in $C \setminus (F_s \cup \{f\})$, it has to use an unused variable as otherwise s would also be in C. Thus, S^* has to fulfill the constraints which proofs the statement. □

Lemma 7. *Every cycle cover that does not contain a tour has at least one cycle C that covers a field f exclusively and does not cover a field f'.*

Proof. Every 2-dimensional grid graph has at least four vertices of degree two. The corresponding fields can only be covered by one cycle. As no cycle covers all fields, there has to exist such a cycle C. □

Theorem 29. *An iterated application of the integer program and the separation constraint in Eq. 4.18 in a connected 2-dimensional grid graph ends after a finite number of iterations with an optimal tour.*

Proof. With every iteration (Lemma 7) on a cycle cover that does not contain a tour we exclude at least the current cycle cover but no optimal tour (Lemma 6). The costs of the cycle covers after each iteration have to be less or equal the optimum tour. This limits

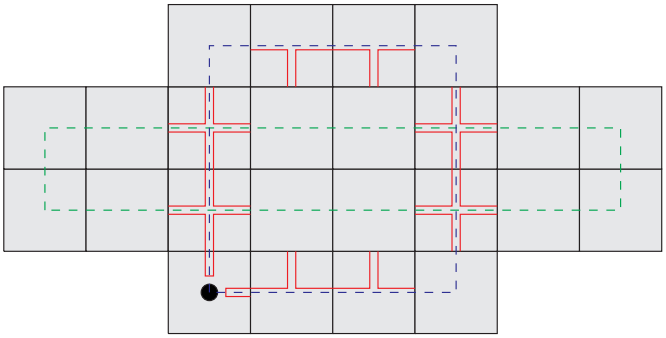


Figure 4.28: The second separation constraint on the blue cycle with the black dotted field as f. At least one of the red passings has to be used.

the number of cycles and also the variations of the cycles to a finite number. After a finite number of iterations, we have excluded all cycle covers with less or equal costs and end with an optimal tour. □

The just described separation constraint is not really difficult. However, it is easier to implement or adapt it wrongly than it might seems.

Formulation 2

We now consider a completely different approach of formulating the integer program. This formulation is based on the alternative problem representation as seen in Sec. 4.2.1. Instead of considering only the connections between adjacent fields, we consider the implicit connections between all fields. Hence, a connection becomes a path that can cover multiple fields and can also contain multiple turns. This needs a quadratic amount of variables while the previous formulation only needed a linear amount. On the other hand, we only need boolean variables instead of integer variables and we know that there are exactly two edges per field used. This eases the subtour elimination and also the adaption to the other variants becomes much simpler than for the other two formulations. This variant has also strong similarities to the classical traveling salesman problem formulation (our subtour separation constraints is nearly identical to a common subtour elimination constraint for the TSP) and common heuristics could be used to help the solver. For example, we could start with a subset of likely edges and iteratively add the remaining edges via column generation. This, however, has not been implemented since even with such additional techniques it is unlikely that the performance of the first formulation is achieved.

We now state the integer programming formulation in detail. Let $G = (V, E)$ be the grid graph. We use an auxiliary graph $G' = (V', E')$ where every field $v \in V$ has four vertices $V'(v) = \{v_{\text{north}}, v_{\text{east}}, v_{\text{south}}, v_{\text{west}}\} \subseteq V'$. Hence, each of these vertices represents the position (field) and the heading. There is an edge $e' = \{v', w'\} \in E'$ for every two vertices $v' \in V'(v), w' \in V'(w), v \neq w \in V$. The cost $c(e')$ represents the shortest connection between v' and the opposite position of w'. E.g. if $v' = v_{\text{north}}$ and $w' = w_{\text{east}}$, then $c(\{v', w'\})$ equals the shortest transition from being at field v heading north to being

at field w heading west. Note that this cost is symmetric thus we could swap v' and w'. The integer program for the cycle cover version can be written as follows:

$$\min \quad \sum_{e' \in E'} c(e') * x_{e'} \tag{4.19}$$

$$\text{s.t.} \quad \sum_{e' \in E'(v_{\text{north}})} x_{e'} = \sum_{e' \in E'(v_{\text{south}})} x_{e'} \quad \forall v \in V \tag{4.20}$$

$$\sum_{e' \in E'(v_{\text{east}})} x_{e'} = \sum_{e' \in E'(v_{\text{west}})} x_{e'} \quad \forall v \in V \tag{4.21}$$

$$\sum_{e' \in E'(v_{\text{east}})} x_{e'} + \sum_{e' \in E'(v_{\text{north}})} x_{e'} = 1 \quad \forall v \in V \tag{4.22}$$

$$x_{e'} \in \mathbb{B} \quad \forall e' \in E' \tag{4.23}$$

Eq. 4.20 and Eq. 4.21 state that if and only if there is an incoming edge on one side there have to be an outgoing edge on the opposite site. Eq. 4.22 states that there have to be exactly two edges entering/leaving (using the symmetry induced by the previous two equations). $E'(v)$ denotes the set of all edges in E' incident to the vertex v.

If we want to obtain an optimal tour and not just a cycle cover, we have to add a subtour separation method. This is simple for this formulation. Assume we have a subtour $C \subsetneq V, C \neq \emptyset$ in our solution. Then we can simply add the constraint that there have to be additional edges between C and $V \setminus C$. This generalizes to the following equation:

$$\sum_{e' \in E'(V'(C), V' \setminus V'(C))} x_{e'} \geq 2 \quad \forall C \subsetneq V, C \neq \emptyset \tag{4.24}$$

The correctness and completeness of this separation constraint should be obvious as it is nearly identical to the traveling salesman problem. The constraint (Eq. 4.24) could have also been stated differently as in Eq. 4.25, being mathematical equivalent but in some cases (with a correlation to the relative size of C) possibly faster in practice, see the work of Pferschy and Stanek [75].

$$\sum_{e' \in E'(V'(C), V'(C))} x_{e'} \leq |V'(C)| - 1 \quad \forall C \subsetneq V, C \neq \emptyset \tag{4.25}$$

Formulation 3

The problem can also be transformed into a variation of the generalized asymmetric TSP where each field has a set of vertices of which at least one has to be visited and the edges are directed. Every field gets a vertex for each of the four directions. Between these four vertices, transitions at the cost of the corresponding turns are possible. Otherwise, there is only an edge going from a vertex to vertex of the same direction of the field neighbored into this direction. This is also displayed in Fig. 4.29.

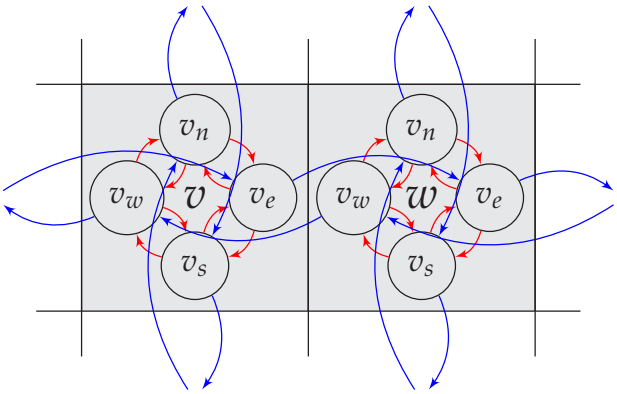


Figure 4.29: Every field has a vertex for each direction of which at least one has to be visited by the tour. Between these vertices are edges with the cost of the corresponding turns. The blue edges are the allowed direct transitions to a neighbor field without a turn.

The auxiliary graph $G' = (V', E')$ is defined as follows:

$$V' = \{v_n, v_e, v_s, v_w \mid v \in V\} \tag{4.26}$$

$$E'_i = \{(v_n, v_e), (v_e, v_n), (v_e, v_s), (v_s, v_e), (v_s, v_w), (v_w, v_s), (v_w, v_n) \mid v \in V\} \tag{4.27}$$

$$E'_e = \{(v_n, w_n), (w_s, v_s) \mid \{v, w\} \in E, v.y < w.y\} \tag{4.28}$$
$$\cup \{(v_e, w_e), (w_w, v_w) \mid \{v, w\} \in E, v.x < w.x\}$$

$$E' = E'_i \cup E'_e \tag{4.29}$$

The costs of the edges are defined by c, where d is the relative distance cost assuming a cost of one per simple turn.

$$c : E' \to \mathbb{R}_0^+, e \mapsto \begin{cases} 1 & e \in E'_i \\ d & \text{otherwise} \end{cases} \tag{4.30}$$

This results in the following base formulation (without subtour elimination):

$$\min \quad \sum_{e \in E'} c(e) * e \tag{4.31}$$

$$\text{s.t.} \quad \sum_{e \in E'^+_e(\{v_n, v_e, v_s, v_w\})} e \geq 1 \quad \forall v \in V \tag{4.32}$$

$$\sum_{e \in E'^+(\{v\})} e = \sum_{e' \in E'^-(\{v\})} e' \quad \forall v \in V' \tag{4.33}$$

$$e \in \mathbb{N}_0 \quad \forall e \in E' \tag{4.34}$$

The constraint in Eq. 4.32 enforces that for every field at least one of its four vertices in V' is visited, i.e., one of the external edges is used. The constraint in Eq. 4.33 enforces every vertex in V' to be entered as often as left. This base formulation provides a cycle cover. For obtaining a tour, we again have to iteratively add subtour separation constraints. $E'^+(W)$

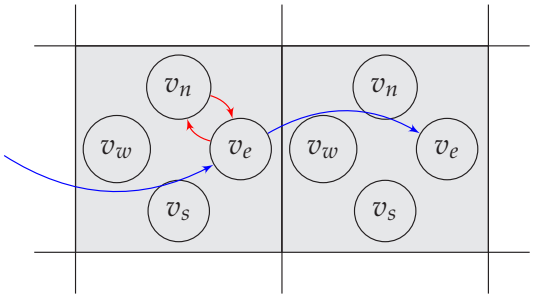


Figure 4.30: Instead of prohibiting the cycle, a simple (+90,-90) can be made. One might think about enforcing to use one of the external instead of the internal edges but then we would effectively have the first (insufficient) subtour constraint.

resp. $E'^{-}(W)$ denotes the set of all (not internal) outgoing resp. incoming edges from the vertex set W.

The application of the first subtour separation constraint of the first formulation to this formulation is obvious. The second subtour constraint is more tricky since now the subtours also have a direction and might be connected or unconnected merely based on their directions. We apply a simpler version of this constraint: For every not spanning subtour that covers one field exclusively, a not used edge in one of its fields has to be used. It can easily be implemented by calculating the strong components (isolated vertices have to be excluded) in the solution graph and check if the corresponding set of fields is not complete and contains at least one field not in the set of any other strong component. The correctness can be deduced similar as in Lemma 6 by the fact that the exclusively covered field somehow has to be connected to the rest. This version is obviously less effective as it contains more edges and excludes fewer possibilities than the original version. However, even a full adaption of the original version might have problems on this IP-formulation as can be seen in Fig. 4.30. If the constraint is also added on the other cycles we want to connect to, the merging should be cheaper since the cycles can share these loops. In the experiments it showed that in most instances the cycle can be connected with no additional turn costs, solving this problem anyway in many cases.

4.4.2 Subset coverage

The adaption of the integer programming formulations for full coverage to subset coverage is simple. For each formulation, we simply enforce the constraints that a field has to be visited only to the fields in the subset. For the second formulation in Sec. 4.4.1 the adaption is the simplest: simply completely remove those fields that are not in the subset. These fields still can implicitly be used by the edges that pass them but there is no need for an explicit visitation. For cycle cover, an additional edge for every field has to be introduced going, e.g., from the west subvertex to the east subvertex with the cost of a minimum cycle. Otherwise, we would, e.g., prohibit cycles that only cover a single field in the subset (but covers some further fields outside the subset). Even the subtour separation constraints remain exactly the same. In case that the subset is relatively small while the

overall number of fields is high, this formulation can actually be in advantage to the other two formulations.

The adaption of the first and third formulations in Sec. 4.4.1 needs a bit more care. First, the $\forall v \in V$ in Eq. 4.14 resp. Eq. 4.32 obviously has to become $\forall v \in S$. This is actually already sufficient for the cycle cover problem. For the tour version, we also have to adapt the subtour separation constraints. For the first (simple) constraint, we have to ensure only to apply it if $C \subsetneq V$ fulfills $C \cap S \neq \emptyset$ and $S \setminus C \neq \emptyset$ (there have to be fields of the subset inside and outside C). For the second separation constraint, the field f has to be in S. We now can get the problem that there might be no field f that is only covered by a single cycle (or at least our previous argument does not work anymore). In this case we can choose f to be in as few cycles as possible and assume these cycles to be connected. It can be easily seen that correctness and completeness is still obtained.

There might be further change in the importance of the first separation constraint. Assume S_1 and S_2 are two distant clusters of S with an expensive connection while there is a lot of space around each cluster. The second separation constraint may only lead to slightly varying cycle covers but has no effective means to actually force a connection (it is still sufficient but the amount of iterations will often be impractical). The first one, in contrast, can force a cycle cover to actually grow by always using the cycle cover of S_1 or S_2 as C. The connections of the resulting increased cycle cover can still be effectively be done by the second one. For such instances, it might be useful to calculate the Voronoi cells with the cycles of a solution as seeds and apply the first separation constraint on all Voronoi cells. This, however, has neither been implemented nor tested in this thesis.

4.4.3 Penalty coverage

The adaption of the integer programming formulations for penalty coverage is more advanced. The cycle cover variants are still straightforward. We only have to express the penalty that is demanded if a field is not visited. In the second formulation in Sec. 4.4.1, this can be done directly because a field is either visited once explicitly or the penalty is paid. For the first and third formulations in Sec. 4.4.1, a field can be visited multiple times and we must prevent that the penalty can be refunded multiple times. This can be done with an auxiliary boolean variable that reflects if a field is visited or the penalty has been chosen.

The tour variants are the actual tricky point: a cycle that has been a subtour in one iteration can be a tour in the next by paying the penalty for the exterior fields. Hence, the subtour separation constraints must have a mechanism to 'turn off' if this happens. Because we actually not explicitly prohibit a subtour but rather enforce that there has to be a change that connects the subtour to the other cycles, we also need to take care of the case that we pay the penalty for all fields in the subtour. Otherwise, we would enforce a connection for the exterior cycles to fields that no longer have to be covered.

We now first consider the adaption for the first (and implicitly third) formulation in Sec. 4.4.1. Afterward, the adaption for the second formulation in Sec. 4.4.1 which can be done simpler.

Adaption of first and third formulation

A field is visited if Eq. 4.14 (resp. Eq. 4.32) is fulfilled otherwise a penalty has to be paid. For simplicity, we now only concentrate on the first formulation as the different notations would probably only induce confusion and the adaption is simple. The problem is that this constraint is an inequality and the left-hand side can take values up to four (assuming a non-zero distance cost). This makes an implementation similar to a Langrangian relaxation (see next section) difficult since we must prevent a negative penalty if a field is visited multiple times (for the second formulation this works). Instead, we introduce for each field v an additional boolean variable x_v that is true if Eq. 4.14 is fulfilled

$$\sum_{\{u,w\}\subseteq N(v)} x_{v,\{u,w\}} \geq 1 * x_v$$

and otherwise it adds the penalty to the objective function

$$(1 - x_v) * \rho(v)$$

This results in the following integer programming formulation for penalty cycle cover:

$$\begin{aligned}
\min \quad & \sum_{v\in V} \sum_{\{u,w\}\subseteq N(v)} c(x_{v,\{u,w\}}) * x_{v,\{u,w\}} \\
& + \sum_{v\in V} (1 - x_v) * \rho(v) && && (4.35)\\
\text{s.t.} \quad & \sum_{\{u,w\}\subseteq N(v)} x_{v,\{u,w\}} \geq x_v && \forall v \in V && (4.36)\\
& 2 * x_{v,\{w\}} + \sum_{u\in N(v),u\neq w} x_{v,\{w,u\}} = \\
& \qquad 2 * x_{w,\{v\}} + \sum_{u\in N(w),u\neq v} x_{w,\{u,v\}} && \forall \{v,w\} \in E && (4.37)\\
& \sum_{\{u,w\}\subseteq N(v)} x_{v,\{u,w\}} \leq 4 * x_v && v \in V && (4.38)\\
& x_{v,\{u,w\}} \in \mathbb{N}_0 && \forall v \in V, \{u,w\} \subseteq N(v) && (4.39)\\
& x_v \in \mathbb{B} && \forall v \in V && (4.40)
\end{aligned}$$

The Eq. 4.38 ensures that if the penalty for a field is paid, the field is no longer covered by any cycle in order to prevent problems during the subtour separation. For the cycle cover version, it can be left out.

We now consider how to adapt the two separation constraints. The technique is similar for both and is based on (partially) implementing a conjunction of the three cases that can happen. For the first (simple) subtour separation constraint we can state these cases as follows

1. The subtour C is to be 'connected' to the other cycles: $\sum_{v\in V\setminus C, w\in C, w'\in V} x_{v,\{w,w'\}} \geq 1$. Note that 'connect' is a rather imprecise term here because C might be heavily deformed in the resulting solution such that it is no longer recognizable.

2. The penalty of all fields in $V \setminus C$ is paid and C becomes a valid tour: $\sum_{v \in V \setminus C} x_v = 0$ or $\sum_{v \in V \setminus C}(1 - x_v) = |V \setminus C|$. For any $S \subseteq V \setminus C$, it also holds that $\sum_{v \in S}(1 - x_v) = |S|$.

3. The penalty of all field in C is paid and C disappears in all optimal solutions: $\sum_{v \in C} x_v = 0$ or $\sum_{v \in C}(1 - x_v) = |C|$. For any $S \subseteq C$, it also holds that $\sum_{v \in S}(1 - x_v) = |S|$.

We can implement this separation constraint by letting the fulfillment of the second or third case make the constraint trivially fulfilled.

$$\sum_{v \in V \setminus C, w \in C, w' \in V} x_{v,\{w,w'\}} \geq 1 - \frac{1}{|V \setminus C|} \left(\sum_{v \in V \setminus C} (1 - x_v) \right) - \frac{1}{|C|} \left(\sum_{v \in C} (1 - x_v) \right)$$

Obviously, this constraint is also fulfilled if the last two cases are only half fulfilled. Hence, the fields for which the penalty has already been paid should be excluded from $V \setminus C$. Let P be the set of fields the current solution has paid the penalty for (or any set with $C \setminus P \neq \emptyset$ and $V \setminus (C \cup P) \neq \emptyset$). If C is a cycle, $C \cap P = \emptyset$ holds by Eq. 4.38. We reformulate the separation constraint as follows:

$$\begin{aligned} \sum_{v \in V \setminus C, w \in C, w' \in V} x_{v,\{w,w'\}} \geq 1 &- \frac{1}{|(V \setminus C) \setminus P|} \left(\sum_{v \in (V \setminus C) \setminus P} (1 - x_v) \right) \\ &- \frac{1}{|C \setminus P|} \left(\sum_{v \in C \setminus P} (1 - x_v) \right) \end{aligned} \tag{4.41}$$

Note that replacing $(V \setminus C) \setminus P$ or $C \setminus P$ by a corresponding subset does not harm the correctness but only makes the 'neutralization' of the constraint more sensitive. Because it also reduces the number of non-zero entries, this can actually be useful in practice.

We can use the same technique for the second subtour separation constraint as both are ≥ 1 inequalities. However, one thing that has to be considered is that this method only focuses on covering a specific field f. If the penalty for f is paid, the constraint has to be neutralized. Hence, this f should have a high penalty. Eq. 4.38 makes sure that only paying the penalty but still covering f is not allowed. This f actually can also be a set, for example all fields in a cycle for which the penalty has not been paid and that are exclusively covered by this cycle. It may also be useful only to select fields with high penalties. Let S be the set of selected fields (may it be a single field as in Sec. 4.4.1 or a set of fields with high penalty) and P the set of fields the penalty has been paid for. F_s are again those fields that are covered by C only in straight passings excluding the fields in S. We can state the separation constraint for a subtour C as follows:

$$\begin{aligned} \sum_{f \in S \wedge \{u,w\} \in N(f), x'_{f,\{u,w\}}=0} x_{f,\{u,w\}} + \sum_{x_t \in T(v), v \in F_s} x_t + \sum_{v \in C \setminus (F_s \cup S), u \neq w \in V, x'_{v,\{u,w\}}=0} x_{v,\{u,w\}} \\ \geq 1 - \frac{1}{|V \setminus (S \cup P)|} \left(\sum_{v \in V \setminus (S \cup P)} (1 - x_v) \right) - \frac{1}{|S|} \left(\sum_{v \in S} (1 - x_v) \right) \end{aligned} \tag{4.42}$$

Obviously, $S \subseteq C \setminus P$, $S \neq \emptyset$ and all fields in S are only in the subtour C. Because this separation can be applied on all subtours and excludes the current solution, it is complete. The correctness is obvious as the constraint is a relaxation of the conjunction of the three cases. In a practical implementation it should be considered that there can be rounding errors and if the sum in case two or case three is evaluated to only close to one and not exactly one, the constraint fails. Solver like CPLEX tolerate some small discrepancy but as a precaution we increase these sums by a small factor in practice.

Formulation 2

The adaption for penalty cycle covers of the second formulation is again very simple. For the adaption of penalty tours, we use the same technique as above. In contrast to the other two formulations, we do not need additional variables for implementing the penalty. First we relax Eq. 4.22 to be ≤ 1, allowing a field to be visited once or not at all. Keeping Eq. 4.20 and Eq. 4.21, we know that a field $v \in V$ is visited if and only if $\sum_{e' \in E'(v_{\text{east}})} x_{e'} + \sum_{e' \in E'(v_{\text{north}})} x_{e'}$ is 1 (otherwise the equation is zero). Hence, the penalty can be expressed as

$$\left(1 - \left(\sum_{e' \in E'(v_{\text{east}})} x_{e'} + \sum_{e' \in E'(v_{\text{north}})} x_{e'}\right)\right) * \rho(v)$$

and simply added to the objective function. Note that this is somehow similar to a Lagrangian relaxation of Eq. 4.22. We can also directly charge the penalty (or its refund) to the edge costs, see Eq. 4.43.

The corresponding integer program can be formally stated as follows

$$\min \quad \sum_{v \in V} \rho(v) + \sum_{e'=\{u',v'\} \in E'} \left(c(e') - \frac{\rho(V(u')) + \rho(V(v'))}{2}\right) * x_{e'} \tag{4.43}$$

$$\text{s.t.} \quad \sum_{e' \in E'(v_{\text{north}})} x_{e'} = \sum_{e' \in E'(v_{\text{south}})} x_{e'} \qquad \forall v \in V \tag{4.44}$$

$$\sum_{e' \in E'(v_{\text{east}})} x_{e'} = \sum_{e' \in E'(v_{\text{west}})} x_{e'} \qquad \forall v \in V \tag{4.45}$$

$$\sum_{e' \in E'(v_{\text{east}})} x_{e'} + \sum_{e' \in E'(v_{\text{north}})} x_{e'} \leq 1 \qquad \forall v \in V \tag{4.46}$$

$$x_{e'} \in \mathbb{B} \qquad \forall e' \in E' \tag{4.47}$$

$V(v')$ maps a vertex $v' \in V'$ to the original field $v \in V$, i.e., $v' \in V'(v)$.

If we want to obtain a penalty tour, we again have to introduce a subtour separation constraint. This one uses the same technique as for the other two formulations above but does not need explicit auxiliary variables. Nonetheless, we introduce a 'virtual' auxiliary variables y_v for all $v \in V$ for readability. These variables are 1 if the field is not visited and otherwise 0. They can simply be substituted by

$$y_v = 1 - \left(\sum_{e' \in E'(v_{\text{east}})} x_{e'} + \sum_{e' \in E'(v_{\text{north}})} x_{e'}\right)$$

For a subtour $C \subsetneq V, C \neq \emptyset$ we can pose the following constraint

$$\sum_{e' \in E'(V'(C), V' \setminus V'(C))} x_{e'} \geq 2 - 2 * \left(\frac{1}{|C \setminus P|} \sum_{v \in C \setminus P} y_v + \frac{1}{|V \setminus (C \cup P)|} \sum_{v \in V \setminus (C \cup P)} y_v \right) \quad \forall C \subsetneq V, C \neq \emptyset \quad (4.48)$$

where P is the set of fields for which the penalty has been paid (or any set with $C \setminus P \neq \emptyset$ and $V \setminus (C \cup P) \neq \emptyset$).

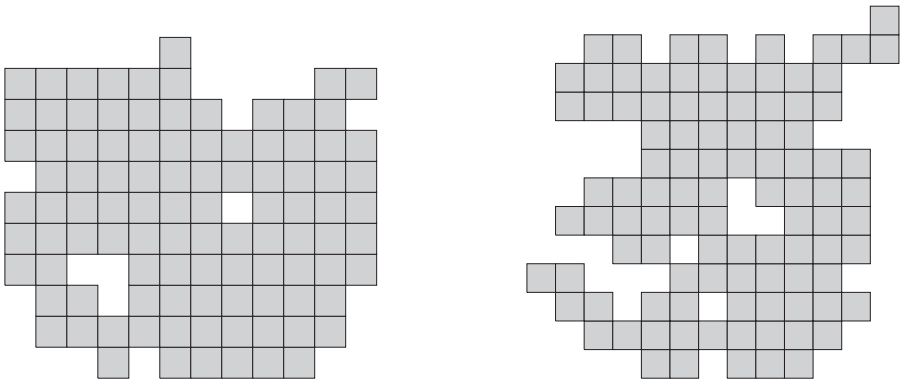
4.5 Experiments

In this section, we provide a first experimental evaluation of the integer programming formulations of the previous section. As we have 24 different formulations and a limited amount of computers and time, we have only tested ten instances per problem size. For the same reason and because the focus is on turn costs, we only consider turn costs but no distance costs, just like de Assis and de Souza [4] and Maurer [7]. Nevertheless, the results give a good intuition of the performance of the different formulations and the difficulty of the problem variants. Despite increased numerical accuracy of CPLEX, there have been slightly inconsistent solutions for subset coverage tours and penalty coverage tours. For the manually checked instances, these could be resolved by a further increment of the numerical accuracy which also increases the runtime. However, this is not reflected by the presented experiments.

4.5.1 Instances

Due to the lack of instances from the real world, we are forced to generate own testing instances. These instances should not just be a random set of fields but be somehow natural. To achieve this, we use dependent probabilities, i.e., the probability if a field is used or not depends on the usage of the neighbored fields. This leads to smoother boundaries and clustering of fields. A random removal of fields as long as they do not destroy the connectivity, in contrast, would lead to a set of thin and curvy paths. They not only seem to be an unrealistic instance for a coverage problem but in first experiments, they seemed also to be easier to solve. However, as these instances have been considered irrelevant, no further experiments have been performed on them.

Our more natural instances are created as follows: First, we create a full square grid with a side-length of $3 * \sqrt{n}$ where n is the desired number of fields. Then we put all existent fields in a list and randomly shuffle it. This shuffled list is now traversed and every field is deleted with a specific probability as long as it does not destroy connectivity. This probability consists of a base probability which is multiplicatively increased for every already missing neighbor fields. Diagonal neighbors have a different (lower) value than the orthogonal neighbors. This is done/repeated until the desired number of fields is reached. Note that a good selection for the size of the initial grid is important for good results with this method.



(a) Dense instance (b) Sparse instance

Figure 4.31: A dense and a sparse instance of size 100 created by the explained method.

We have two sets of parameters to create instances for our experiments. The first creates rather dense grids with smooth boundaries and few 'pseudopodia' by having a low base probability and high orthogonal and diagonal neighbor values. The second creates rather sparse grids with more disturbed boundaries, more small holes, and more 'pseudopodia' by using a slightly higher base probability and lower orthogonal and diagonal neighbor values. Examples of these two types for $n = 100$ can be seen in Fig. 4.31.

To create a subset for subset coverage instances we put all fields in a list, randomly shuffle it, and then take the first m elements where m is the desired size of the subset. For our experiments, we select m to be 25% of the fields. The penalties for penalty coverage a selected by a normal distribution with a mean of 1 and a standard deviation of 1. If the value is negative, it is set to zero. Further, we round the values to two decimal places to prevent numerical instabilities. Dependent probabilities to create subsets and penalties have not been used in experiments but are planned for future experiments.

The used instances are provided on the thesis' website. Due to a bug in the instance generator, the sequence of subset selection and penalty values is identical for all nearly all instances. This does not invalidate the experimental results because the assignment of the values to the fields is dependent on the grid graph which, however, differs. Still, the instances are not as random as intended and new experiments (also with more instances per data point) should be performed to verify the results.

4.5.2 Results

The experiments were executed on modern 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 12.5.0.0 with the parameters `EpInt`=0, `EpGap`=0, `EpOpt`=1×10^{-9}, and `EpAGap`=0. No further optimizations have been performed.

The percentages of instances solved to optimality within 15 min are plotted in Fig. 4.32 for full cycle cover resp. in Fig. 4.33 for full tour. It can be seen that for cycle cover the third formulation performs best and can solve instances of a size of up to 1000-1500 fields

with a success ratio of around 50%. The first formulation can also solve some instances, the third formulation could not but remains visibly less efficient. As expected, the second formulation performs worst and only is able to solve 50% of the instance up to a size of 400-500 fields. It is also able to solve one instance of size 2900 but it seems like the corresponding instance is surprisingly easy to solve by all solvers. For full tours, the first formulation is able to beat the third formulation slightly and able to solve 50% of the instances with a size of around 700 fields. The additional usage of the first separation constraint results in some instance becoming solvable in time and other becoming unsolvable in time. The second formulation is only able to solve 50% of the instances with 300-400 fields but also the previously mentioned large instance. The runtime is plotted in Fig. 4.34 resp. Fig. 4.35. It can be seen that the third formulation seems to be slightly slower that the first formulation, in especially if it uses both separation constraints. As we cut off the runtime at 15 min, the plot is distorted because for the larger instance sizes only the easy instances are counted with their real runtime and should only be evaluated in combination with the previous two plots.

For subset coverage, the percentage of instances solved within 15 min are plotted in Fig. 4.36 for cycle cover and in Fig. 4.37 for tour. The first formulation performs best for the cycle cover and for tour problem. Using additionally the first separation constraint improves the performance already for the randomly selected subset fields. The third formulation performs even worse than the second. The runtime is plotted in Fig. 4.38 resp. Fig. 4.39. For a few instances, the first and the third formulation threw errors during the solution process. For most of these 'degenerated instances', previously numerical problems for the second formulation have been observed, i.e., the returned solutions were not optimal (observed by resolving with the 'numerical emphasize' parameter which led to a better solution). The reason for this has not been discovered within the period of the thesis.

For penalty coverage, the percentage of instances solved within 15 min are plotted in Fig. 4.40 for cycle cover and in Fig. 4.41 for tour. The corresponding average runtime can be seen in Fig. 4.42 resp. Fig. 4.43. The first formulation performs the best and can surprisingly solve at least 40% of all instance sizes within the time limit. Hence, the performance seems to be better than for the full coverage variants. Using both separation constraints improves the performance slightly. However, there has been at least one instance with numerical problems. This instance was the only one in our whole experiments for which the solutions of the different formulations did not coincide.

In the evaluation of the individual instances, it has been noticed that the *sparse* instances are significantly harder to solve than the *dense* instances. This is unexpected because the sparse instances have more trivial boundary fields than the dense instances. Furthermore, the cost of the tour was for all problem variants very often equal to the cost of the cycle cover or at least only slightly higher. A reason for this could be the rather rough boundaries of the instances and the missing distance costs. Additional experiments with different instances and distance costs should be made for further evaluations.

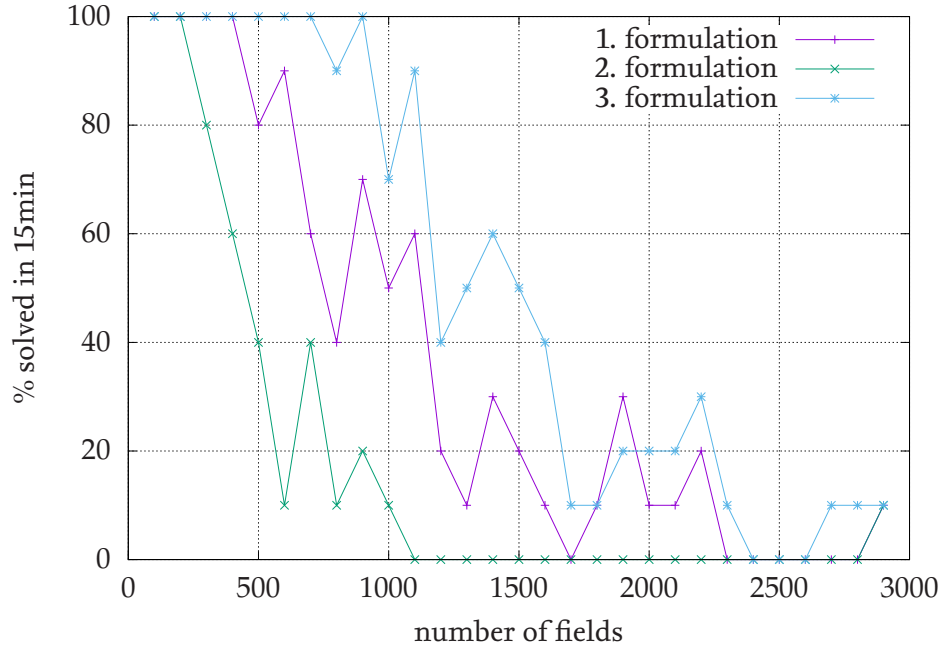


Figure 4.32: Percentage of instances solved for full coverage cycle cover. The instances have a size of $100, 200, \ldots 2900$ fields and there are five dense and five sparse instances per size. The cost weights are $\tau = 1, \kappa = 0$.

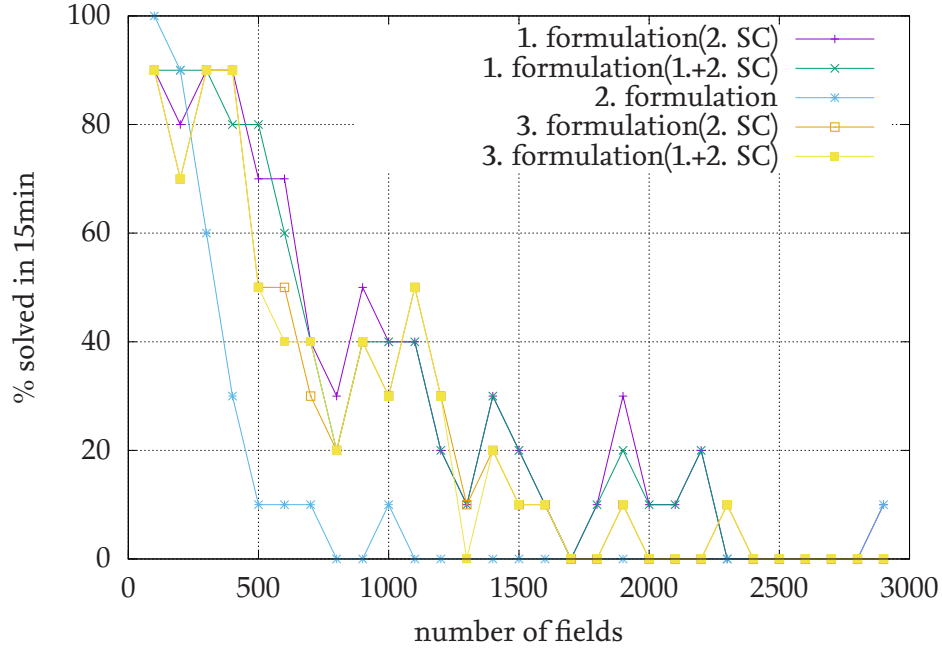


Figure 4.33: Percentage of instances solved for full coverage tours. The instances have a size of $100, 200, \ldots 2900$ fields and there are five dense and five sparse instances per size. The cost weights are $\tau = 1, \kappa = 0$.

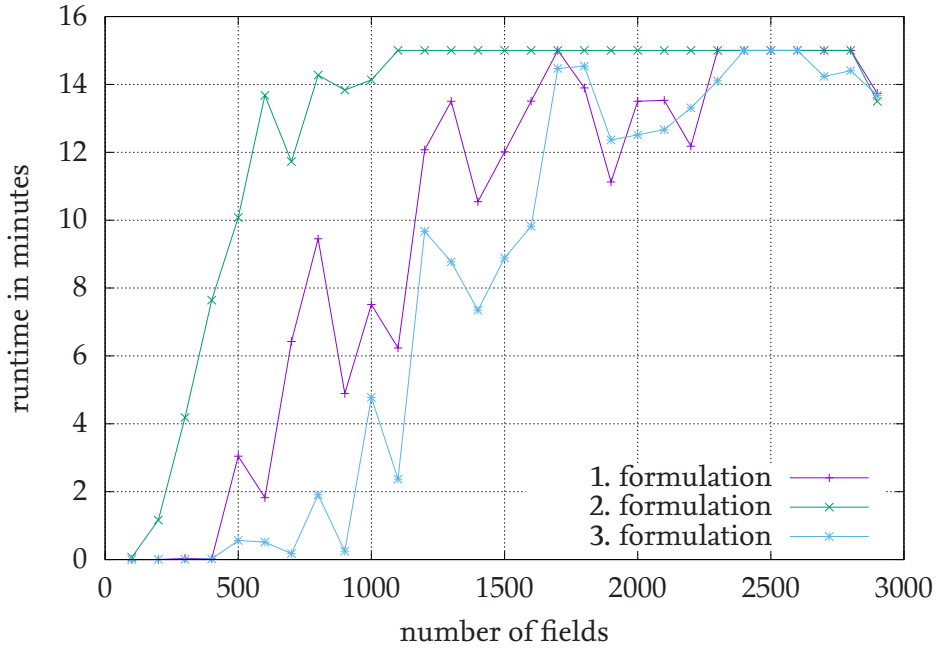


Figure 4.34: Average runtime for full coverage cycle cover with a cutoff at 15 min. The instances have a size of 100, 200, ... 1400 fields and there are five dense and five sparse instances per size. The cost weights are $\tau = 1, \kappa = 0$.

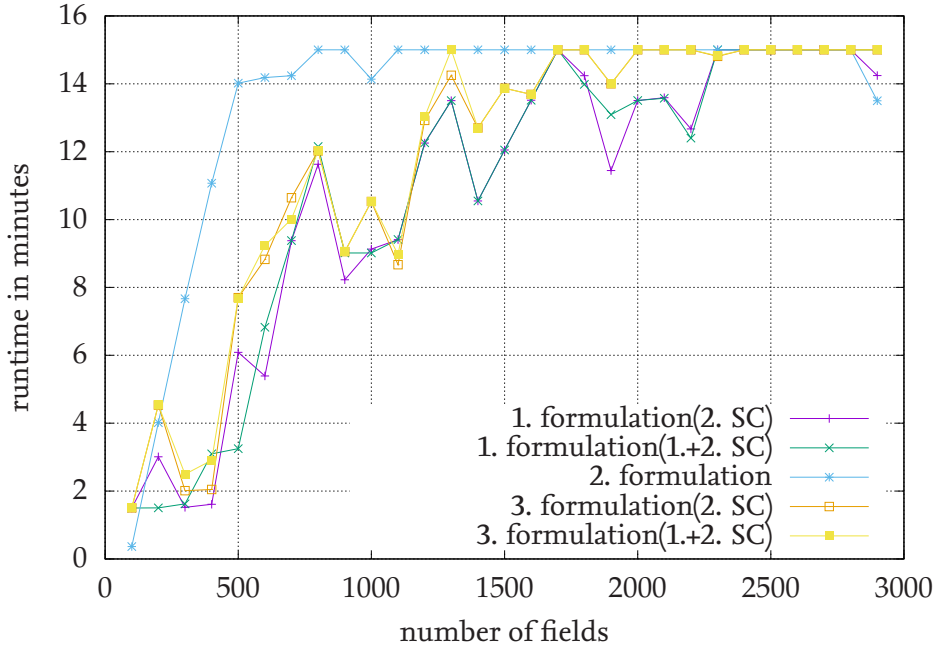


Figure 4.35: Average runtime for full coverage tour with a cutoff at 15 min. The instances have a size of 100, 200, ... 1400 fields and there are five dense and five sparse instances per size. The cost weights are $\tau = 1, \kappa = 0$.

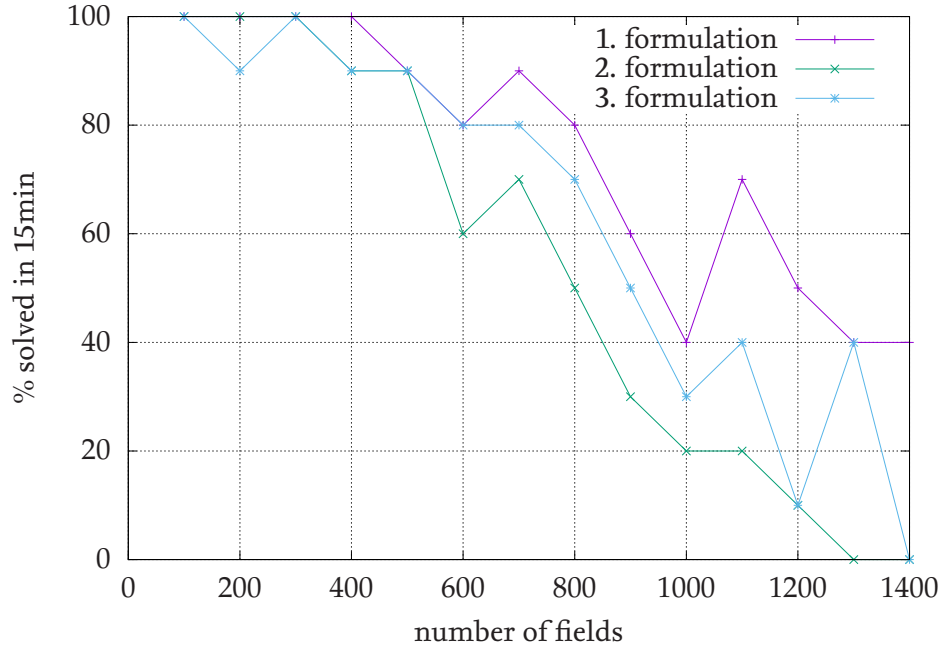


Figure 4.36: Percentage of instances solved for subset coverage cycle cover. The instances have a size of $100, 200, \ldots 1400$ fields and there are five dense and five sparse instances per size with 25% randomly chosen subset fields. The cost weights are $\tau = 1, \kappa = 0$.

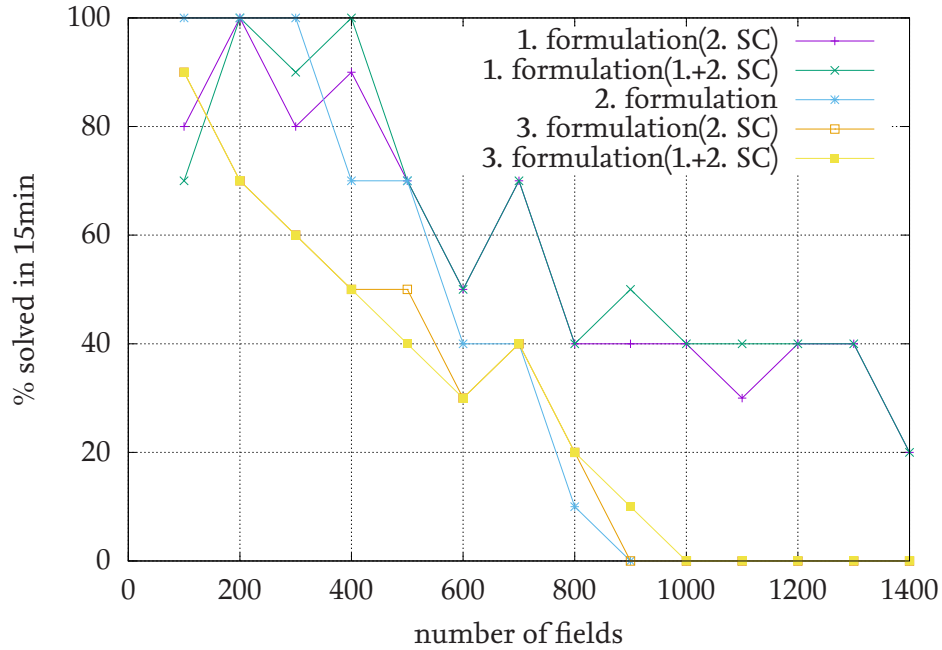


Figure 4.37: Percentage of instances solved for subset tours. The instances have a size of $100, 200, \ldots 1400$ fields and there are five dense and five sparse instances per size with 25% randomly chosen subset fields. The cost weights are $\tau = 1, \kappa = 0$.

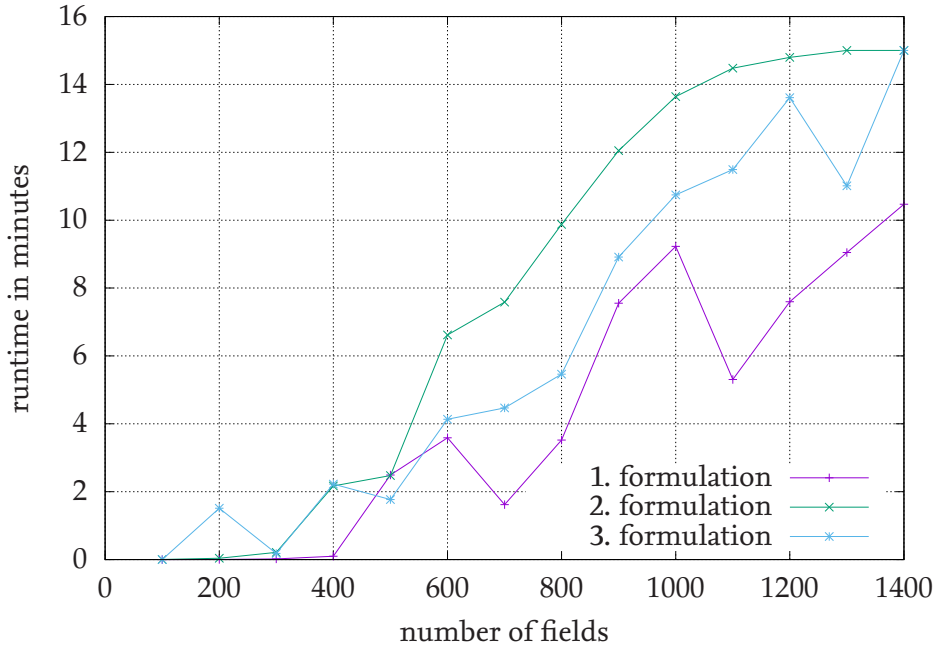


Figure 4.38: Average runtime for subset coverage cycle cover with a cutoff at 15 min. The instances have a size of 100, 200, . . . 1400 fields and there are five dense and five sparse instances per size with 25% randomly chosen subset fields. The cost weights are $\tau = 1, \kappa = 0$.

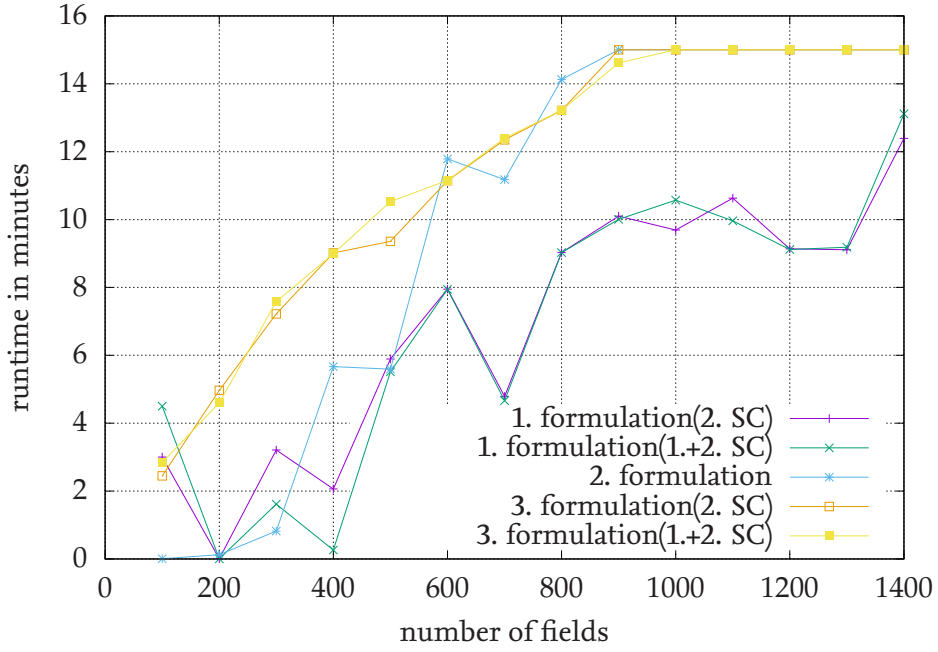


Figure 4.39: Average runtime for subset tours with a cutoff at 15 min. The instances have a size of 100, 200, . . . 1400 fields and there are five dense and five sparse instances per size with 25% randomly chosen subset fields. The cost weights are $\tau = 1, \kappa = 0$.

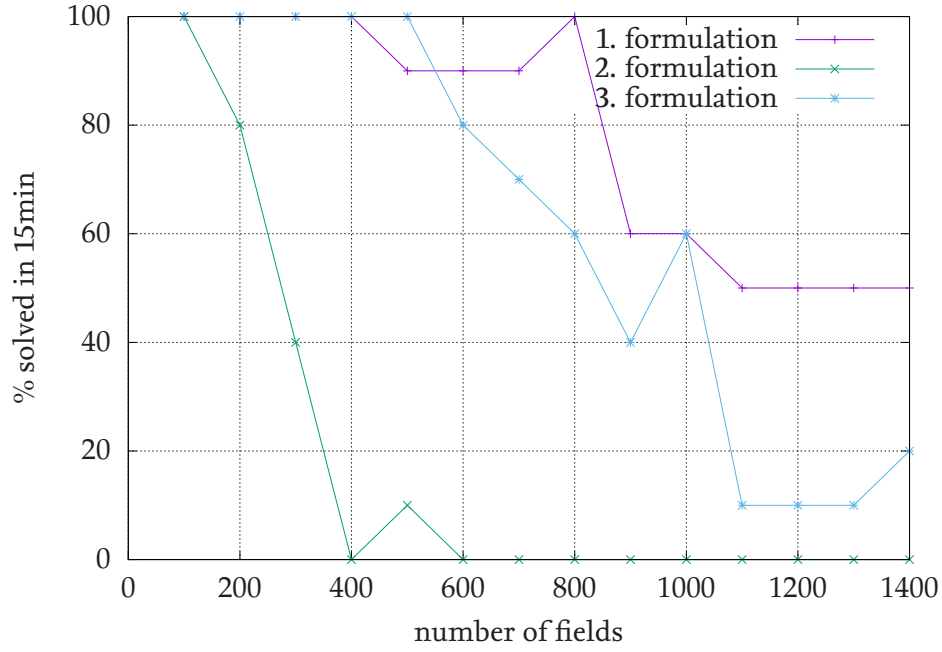


Figure 4.40: Percentage of instances solved for penalty coverage cycle cover. The instances have a size of $100, 200, \ldots 1400$ fields and there are five dense and five sparse instances per size. The cost weights are $\tau = 1, \kappa = 0$.

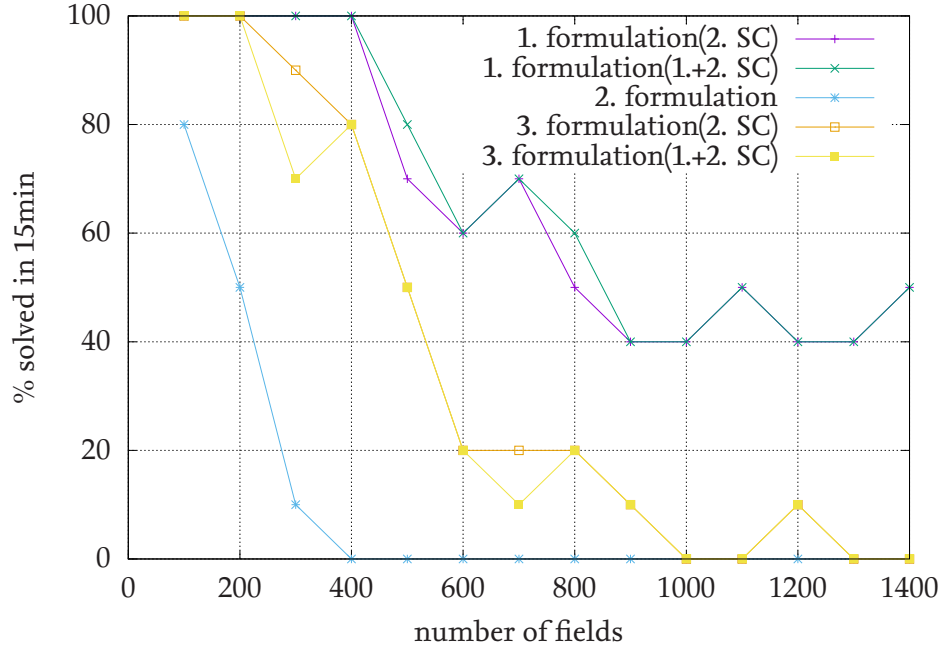


Figure 4.41: Percentage of instances solved for penalty tours. The instances have a size of $100, 200, \ldots 1400$ fields and there are five dense and five sparse instances per size. The cost weights are $\tau = 1, \kappa = 0$.

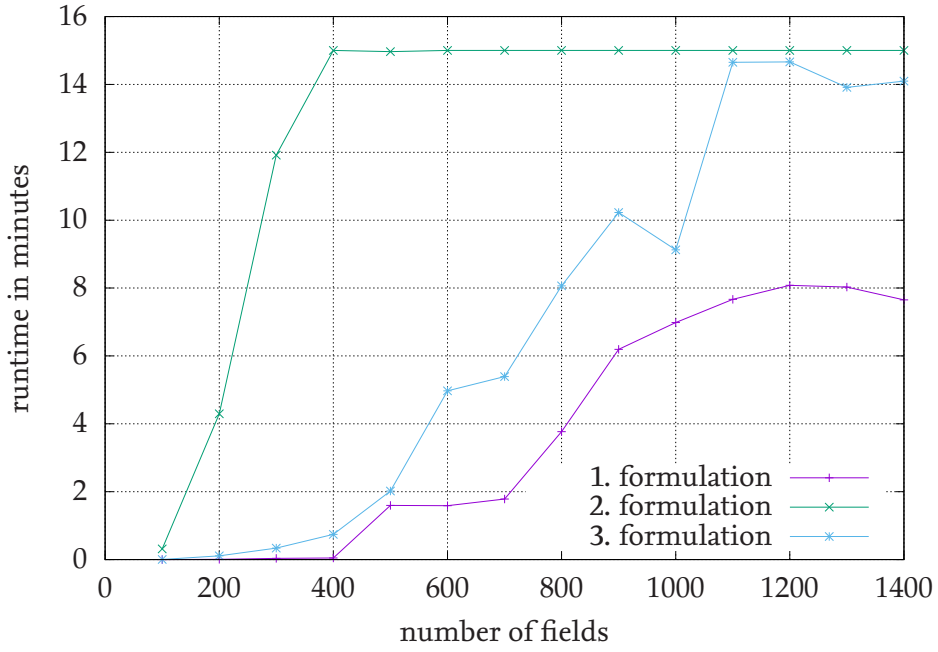


Figure 4.42: Average runtime for penalty coverage cycle cover with a cutoff at 15 min. The instances have a size of 100, 200, ... 1400 fields and there are five dense and five sparse instances per size. The cost weights are $\tau = 1, \kappa = 0$.

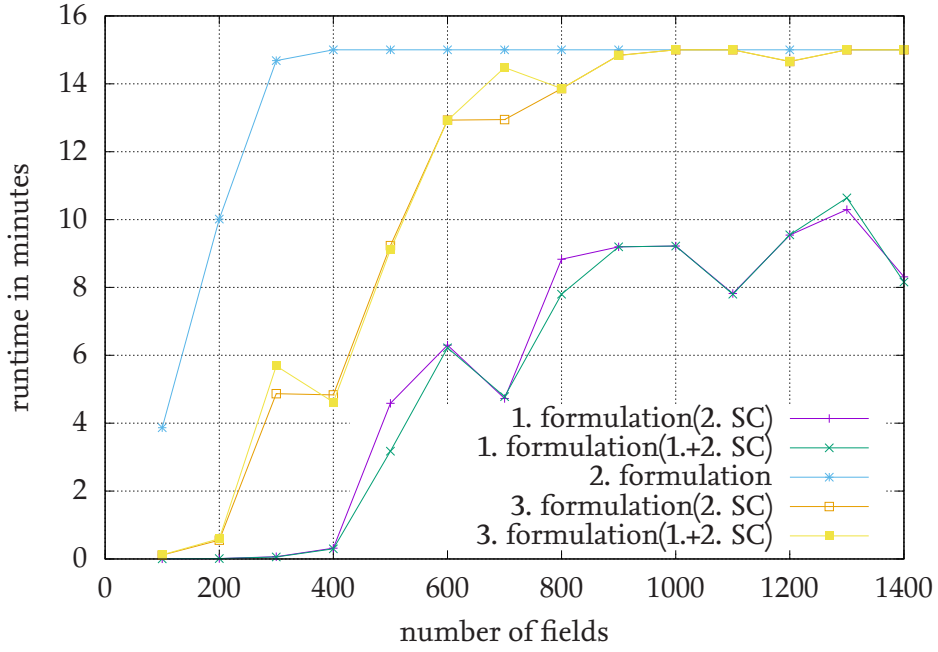


Figure 4.43: Average runtime for penalty tours with a cutoff at 15 min. The instances have a size of 100, 200, ... 1400 fields and there are five dense and five sparse instances per size. The cost weights are $\tau = 1, \kappa = 0$.

5 Geometric instances

In this chapter, we consider the problem of covering a set of points in a 2-dimensional polygonal environment.

Limiting the movement of, e.g., a drone to a grid graph can in some scenarios be too restrictive to be of practical use. This would only allow sharp $\pm 90^\circ$ and 180° turns in a square grid and $\pm 60^\circ$, $\pm 120^\circ$, and 180° in a hexagonal grid. Further, the interesting areas are probably rather small compared to the reachable area. Thus, the grid graph would consist of a small amount of vertices that are actually interesting and a large amount of vertices that are just for the paths. In this chapter, we relax the restriction to the grid graph and allow arbitrary movements around polygonal obstacles. The interesting areas are implemented as sets of points that can be passed in a constant amount of fixed orientations. A visualization of this idea can be seen in Fig. 5.1. The reachable room can be easily calculated via the Minkowsky sum of the bounding polygons of obstacles and the bounding polygon of the drone such that the drone can be assumed to be a point. To be safe, it might be recommendable to choose the bounding polygon of the drone slightly larger. More on this can be found in the book of de Berg et al. [14]. The discretization of the passing orientations allows us to apply the approximation method of Sec. 4.3. Without discretization and obstacles, the full coverage geometric cycle cover and tour problem equals the angular metric cycle cover and traveling salesman problem. For these, only $O(\log n)$-approximation algorithms by Aggarwal et al. [1] are known to us. The subset coverage problem is obviously no longer interesting in this problem setting such that we only consider full coverage and penalty coverage.

Assuming a drone can measure its success, the set of interesting points can be updated and the passing orientations can be adapted after each flight. In practice, one would, of course, smooth the turns and not strictly stick with the fixed orientations. The fixed orientations, however, ease the theoretical consideration of the problem and allow us constant factor approximations (only dependent on the constant number of allowed passing orientations). Also larger instances can be solved to optimality than for the quadratic traveling salesman problem (compare [41]) using integer programming. In case we are satisfied with only a single orientation, the cycle cover variant of full coverage becomes optimally solvable in polynomial time. Otherwise, we obtain the same approximation ratios as in a grid graph. A remaining problem is to calculate the cheapest connections between two points around the obstacles. This, however, is nearly as simple as for the euclidean shortest path.

In the next section (Sec. 5.1), we give formal problem descriptions. In Sec. 5.2, we show that the minimum cost path with distance and turning costs in a 2-dimensional polygonal environment between two configurations can be calculated in polynomial time by using the visibility graph. In Sec. 5.3, we generalize the approximation algorithms of Sec. 4.3 to

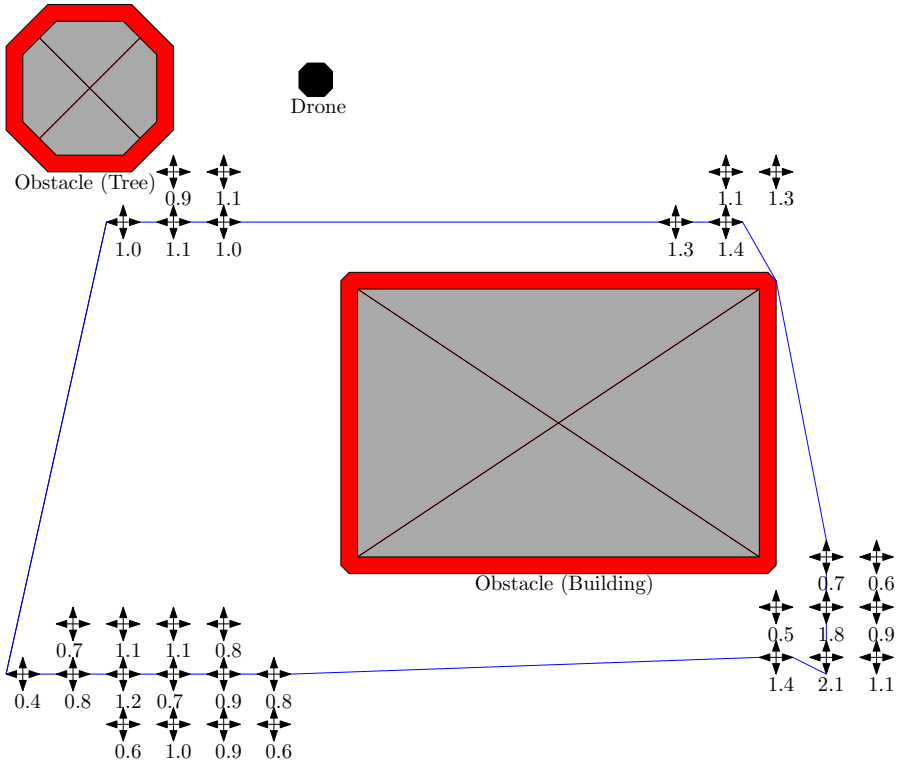


Figure 5.1: We plan a penalty tour one a set of points with given orientations for passing (two in this case). The obstacles (gray) are extended by the Minkowsky sum with the drone in order to assume the drone to be a point. As the drone can measure the mosquitoes killed at a point, the selected points, the orientations, and the expected catching rates can be adapted after each run. The displayed tour is not optimal but only an example.

the geometric variant. Finally, we state an integer programming formulation in Sec. 5.4 and we provide some experimental data in Sec. 5.5.

5.1 Problems

We again first give formal problem descriptions. While the discretization of the passing orientations allows us to provide simple constant factor approximation algorithms (for fixed discretization resolution), the definition of the costs of a cycle becomes more complicated compared to the non-discretized version. We express a cycle as a sequence of configurations (position and heading). A point p with the discretized orientations $\delta(p)$ is covered by a cycle if one of its configurations $\{(p,\alpha),(p,\alpha+\pi) \mid \alpha \in \delta(p)\}$ is in the cycle. The transition between two subsequent such configurations equals the optimal transition and the cost of the cycle is the sum of the cost of these transitions.

Definition 2. *A cycle C for a set of points P with the discrete orientations $\delta : P \to \mathcal{P}([0,\pi))$ in a polygonal environment with the obstacles $\mathcal{O}$ is a sequence of configurations $\sigma_0 = (p_0 \in P, \alpha_0 \in \{\alpha, \alpha+\pi \mid \delta(p_0)\}), \ldots, \sigma_n = (p_n \in P, \alpha_n \in \{\alpha, \alpha+\pi \mid \delta(p_n)\})$ with $\forall i = 0,\ldots,n-1 : p_i \neq p_{i+1}$. The cost $c_{\mathcal{O},\kappa,\tau}(C)$ of a cycle is the sum of minimum cost transitions between any two subsequent configurations:*

$$c_{\mathcal{O},\kappa,\tau}(C) = SP_{\mathcal{O},\kappa,\tau}(c_n, c_0) + \sum_{i=0,\ldots,n-1} SP_{\mathcal{O},\kappa,\tau}(c_i, c_{i+1})$$

where $SP_{\mathcal{O},\kappa,\tau}(c_a, c_b)$ is denoting the minimum cost transition (shortest path) from configuration c_a to c_b around the obstacles $\mathcal{O}$ with the distance cost weight κ and the turn cost weight τ. If no such transition exists, we assume the cost to be infinite.

For a point $p \in P$ and a cycle C, we denote $p \in C \iff \exists \beta \in \{\alpha, \alpha+\pi \mid \alpha \in \delta(p)\} : (p,\beta) \in C$ as p being covered by C.

The calculation of the minimum cost transitions/paths is considered in Sec. 5.2. As every discrete orientation consists of two opposing headings, the cycle is symmetric, i.e., it is identical to the reversed configuration sequence with every heading rotated by 180°. An alternative definition of such a cycle could be to represent every discrete orientation of the vertices as an ϵ-length segment that has to be in the cycle. This definition would be closer to the geometric interpretation of the problem but the formal definition would be more difficult since ϵ-segments cannot be expressed by concrete coordinates.

In the full coverage variant, we want a set of cycles (cycle cover) or a single cycle (tour) such that the sum of distance costs and turn costs are minimal and every point $p \in P$ is covered.

Problem 8 (Full coverage geometric cycle cover)**.** *Given a set of polygonal obstacles $\mathcal{O}$ in $\mathbb{R}^2$, a set of points P that do not lie within any obstacle, and a function $\delta : P \to \mathcal{P}([0,\pi))$ with $\forall p \in P : |\delta(p)| \leq \omega$ for some fixed $\omega \in \mathbb{N}^+$. Further, the distance cost weight $\kappa \in \mathbb{R}_0^+$ and the turn cost weight $\tau \in \mathbb{R}_0^+$. Wanted is a set of cycles $\mathcal{C} = C_0,\ldots,C_m$ that do not intersect $\mathcal{O}$ with $\sum_{C_i \in \mathcal{C}} c_{\mathcal{O},\kappa,\tau}(C_i)$, is minimal, and every point $p \in P$ is covered by a least on cycle $C_i \in \mathcal{C}$.*

Problem 9 (Full coverage geometric tour)**.** *Given a set of polygonal obstacles $\mathcal{O}$ in $\mathbb{R}^2$, a set of points P that do not lie within any obstacle, and a function $\delta : P \to \mathcal{P}([0, \pi))$ with $\forall p \in P : |\delta(p)| \leq \omega$ for some fixed $\omega \in \mathbb{N}^+$. Further, the distance cost weight $\kappa \in \mathbb{R}_0^+$ and the turn cost weight $\tau \in \mathbb{R}_0^+$. Wanted is a single polygonal cycle C that does not intersect $\mathcal{O}$ with $c_{\mathcal{O},\kappa,\tau}(C)$, is minimal, and covers every point $p \in P$.*

In the penalty coverage variant, we want a set of cycles (cycle cover) or a single cycle (tour) such that the sum of distance costs, turn costs, and penalty costs are minimal. The penalty $\rho(p)$ is added for every point $p \in P$ that is not covered.

Problem 10 (Penalty geometric cycle cover)**.** *Given a set of polygonal obstacles $\mathcal{O}$ in $\mathbb{R}^2$, a set of points P that do not lie within any obstacle, a function $\delta : P \to \mathcal{P}([0, \pi))$ with $\forall p \in P : |\delta(p)| \leq \omega$ for some fixed $\omega \in \mathbb{N}^+$, and a penalty function $\rho : P \to \mathbb{R}^+$. Further, the distance cost weight $\kappa \in \mathbb{R}_0^+$ and the turn cost weight $\tau \in \mathbb{R}_0^+$. Wanted is a set of cycles $\mathcal{C}$ with $\sum_{C_i \in \mathcal{C}} c_{\mathcal{O},\kappa,\tau}(C) + \sum_{p \in P, \forall C_i \in \mathcal{C} : p \notin C_i} \rho(p)$ is minimal.*

Problem 11 (Penalty geometric tour)**.** *Given a set of polygonal obstacles $\mathcal{O}$ in $\mathbb{R}^2$, a set of points P that do not lie within any obstacle, a function $\delta : P \to \mathcal{P}([0, \pi))$ with $\forall p \in P : |\delta(p)| \leq \omega$ for some fixed $\omega \in \mathbb{N}^+$, and a penalty function $\rho : P \to \mathbb{R}^+$. Further, the distance cost weight $\kappa \in \mathbb{R}_0^+$ and the turn cost weight $\tau \in \mathbb{R}_0^+$. Wanted is a single polygonal cycle C with $c_{\mathcal{O},\kappa,\tau}(C) + \sum_{p \in P, p \notin C} \rho(p)$ is minimal.*

With obstacles, a grid graph can be modeled and the NP-hardness of the full coverage geometric cycle cover follows from the NP-hardness of the subset coverage cycle cover problem (in 2-dimensional grid graphs) (Theorem 8). As it is contained in all the other problems, all the other problems are NP-hard, too. Even without obstacles, we can use the NP-hardness proof for the angular metric cycle cover problem by Aggarwal et al. [1].

5.2 Geometric shortest paths with turn costs

A primitive in finding a tour with minimum cost is to find the path with minimum cost between two configurations. In the grid graph scenario, we already have a graph abstraction and there is a simple transformation of the graph to solve this problem via a classical shortest path algorithm like Dijkstra's algorithm. In the geometric scenario, there is an infinite amount of possible turning points for a path. For the euclidean shortest path problem in 2-dimensional polygonal environments, the shortest path problem (only euclidean length) can be easily solved by the observation that an optimal path can only make turns at the vertices of the obstacles. This can be easily seen by imagining the shortest path as a rubber band which, of course, would only make turns at points it hits an obstacle. We can simply calculate a visibility graph of the vertices of the obstacles and the two end points and perform a shortest path calculation in this graph. If we do not only have distance costs but also turn costs (linear to the turning angle), we can show that there is a minimum cost path that only has turning points at the end points and the vertices of the obstacles. Hence, we can again simply calculate the visibility graph, transform it to integrate turn costs, and then use Dijkstra's algorithm to obtain a minimum cost path. In

the book of de Berg et al. [14] is a nice chapter on the euclidean shortest path and the following part uses a similar reasoning. Contrary to the euclidean shortest path, a minimum cost path with only turn costs can also have other turning points such as can be seen in Fig. 5.2.

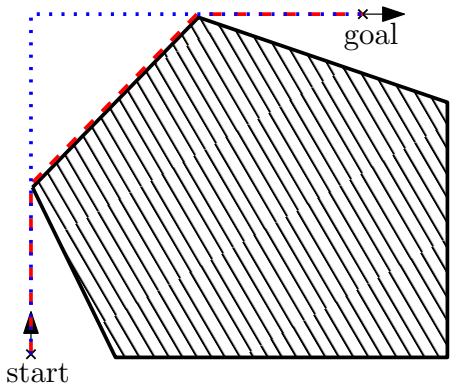


Figure 5.2: The blue and the red path have both minimum turn costs. The red one only has turns on the obstacle vertices, the blue one not.

Let us first define a minimum cost path with distance and turn cost.

Problem 12. *Given a set $\mathcal{O}$ of disjunct polygonal obstacles, a turn cost weight $\tau \in \mathbb{R}_0^+$, and a distance cost weight $\kappa \in \mathbb{R}_0^+$. For a start configuration $(p_s \in \mathbb{R}^2, \alpha_s \in [0, 2\pi))$ and a goal configuration $(p_g \in \mathbb{R}^2, \alpha_g \in [0, 2\pi))$, we want a polygonal path $p_0(= p_s), p_1, \dots p_{n-1}, p_n(= p_g)$ such that*

$$\kappa * \left(\sum_{i=0,\dots,n-1} d(p_i, p_{i+1}) \right) + \tau * \left(ta(p_0 - (\sin(\alpha_s), \cos(\alpha_s)), p_0, p_1) \right.$$
$$\left. + ta(p_{n-1}, p_n, p_n + (\sin(\alpha_g), \cos(\alpha_g))) + \sum_{i=0,\dots,n-2} ta(p_i, p_{i+1}, p_{i+2}) \right)$$

is minimal and the interior of $\mathcal{O}$ is not intersected. $ta(a, b, c)$ denotes the turning angle between segment (a, b) and segment (b, c).

Note that the obstacles are open, i.e., the boundary does not count as interior.

We first show that we can limit the turning points to the obstacle vertices $V(\mathcal{O})$ by showing that if a minimum cost path has a turning point not in $V(\mathcal{O})$, we can modify the path such that it has the same turn costs but a smaller distance cost which would be a contradiction to its optimality. If we have distance costs, all minimum cost paths have only turning points in $V(\mathcal{O})$. If we only have turn costs, at least one path with minimum cost has only turning points in $V(\mathcal{O})$, namely the shortest one.

Theorem 30. *A path with minimum costs ($\kappa > 0$) has only turning points at the vertices of the obstacles.*

Proof. Assume there is no path with minimum cost that has turning points only at the vertices of the obstacles. Then select the shortest path with minimum cost. There is at least one turning point p_i not on a vertex of an obstacle and there is a circle of non-zero radius with the two incident segments (p_{i-1}, p_i) and (p_i, p_{i+1}) of p_i in the path such that the triangle p_i, p_i', p_i'' does not intersect with the obstacles. p_i' and p_i'' are the two intersection points of the two segments with the circle. We can build a shorter tour by

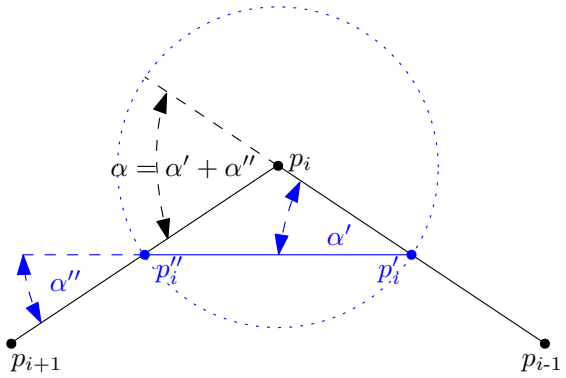


Figure 5.3: If the turning point p_i is not on a vertex of an obstacle, we can build a shorter tour with the same turn costs by replacing p_i by p_i' and p_i'' (blue).

replacing the segments (p_{i-1}, p_i) and (p_i, p_{i+1}) by (p_{i-1}, p_i'), (p_i', p_i''), and (p_i'', p_{i+1}). As can be easily seen in Fig. 5.3, the new path is shorter. Further, it can be seen that the two new turning angles α' and α'' are a partition of the old turning angle α and hence it holds that $\alpha = \alpha' + \alpha''$ and the turn costs do not increase. This is a contradiction to the initial assumption and concludes the proof. □

We now can restrict ourselves to the visibility graph of $V(\mathcal{O}) \cup \{p_s, p_g\}$ to find the path with minimum cost. A naïve way of calculating the visibility graph is to add an edge between two points p and p' if the segment (p, p') does not intersect with any of the $O(n)$ segments of the obstacles. This obviously has a runtime of $O(n^3)$ where n is the number of vertices in the obstacles. There exist worst case optimal algorithms that run in $O(n^2)$ by Asano et al. [56] and Welzl [57] and an algorithm by Ghosh and Mount [58] that runs in $O(n \log n + k)$ where k is the number of edges in the visibility graph. As k is in $O(n^2)$, the later algorithm is also worst case optimal but can achieve a better runtime if the resulting visibility graph is sparse.

With the visibility graph in hand, we only need to integrate the turn costs by a simple transformation before applying a shortest path algorithm for directed weighted graphs such as Dijkstra's algorithm. We have two different kinds of edges: first, edges to transit between two positions with the cost of the euclidean distance, and second, edges to rotate at a specific point into the direction of the next neighbored point with the cost of the rotation. This idea has also been used in the third integer program formulation for grid graphs, see Sec. 4.4.1. While it is rather easy to grasp, e.g., from Fig. 5.4, it is rather complicated to describe formally.

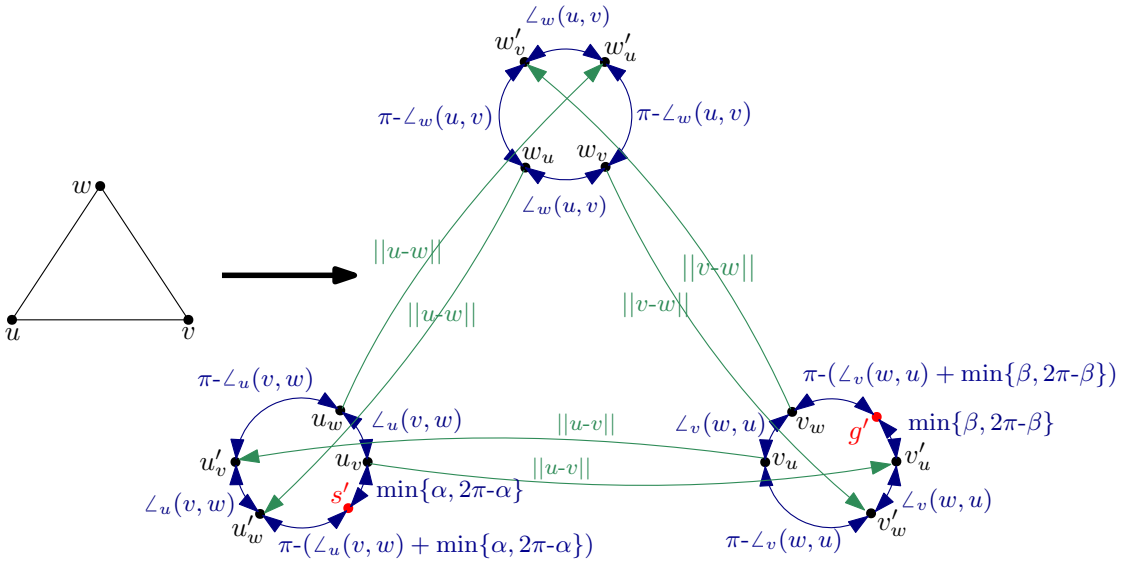


Figure 5.4: The transformation of the graph to integrate turn costs in the edge costs. The start configuration is in u heading south-east and the goal configuration is in v heading north-east. The cost weights κ, τ are ignored. $\angle_a(b,c)$ denotes the counterclockwise angle from segment (a,b) to segment (a,c).

The formal procedure is contained in Algorithm 3. Let $\angle(\overrightarrow{p})$ represent the counterclockwise angle (radian between 0 and 2π) between $\overrightarrow{p}$ and the x-axis. First, the visibility graph on the vertices of the obstacles and the start and goal position is calculated. Then we transform this visibility graph into a new digraph $D(V', E')$ by creating four vertices and two arcs for each edge. The vertices describe the state of leaving through the edge and entering through the edge. An artificial map A is kept to remember the corresponding heading for the newly created vertices. The map P remembers the corresponding point of the state of a newly created vertex. The map I remembers which new vertices have been created for a vertex in the visibility graph. Then, we add the new vertices s' and g' that represent the start and goal states. Afterward, we add internal 'rotation arcs' for every point that represents a heading at this point. The rotation arcs are an undirected cycle for each vertex in the visibility graph. A rotation in a solution, hence, usually consists of multiple small rotations within this cycle. This reduces the number of necessary arcs in the new graph compared to creating an arc for every possible (reasonable) rotation. In this new graph, we can calculate the shortest path between the start state and the goal state. Afterward, we recover the corresponding positions from this shortest path by the map P and eliminate the duplicates that are created by the partial rotations.

Theorem 31. *The shortest path between two configurations in terms of turning angle sum and length along polygonal obstacles $\mathcal{O}$ can be calculated in $O(n^2 * \log n)$ where n is the number of vertices in $\mathcal{O}$.*

Proof. The visibility graph can be calculated in $O(n^2)$. The first loop obviously runs in $O(|E(\mathcal{O})|)$ and adds $O(|E(\mathcal{O})|)$ vertices and edges to the new graph G'. In the operations

Algorithm 3: Minimum cost path in polygonal environments with distance and turn costs

Data: Start position s and heading α, goal position g and heading β, a set of obstacles $\mathcal{O}$, distance cost weight $\kappa \geq 0$, turn cost weight $\tau \geq 0$

Result: A sequence of positions representing the minimum cost path.

Calculate visibility graph $G(V(\mathcal{O}) \cup \{s, g\}, E)$;
Set default value for all uninitialized variables/maps to $\emptyset$;
`/* Transform edges of visibility graph */`
for $e = \{u, v\} \in E$ **do**
  $V' \leftarrow V' \cup \{u_v, u'_v, v_u, v'_u\}$;
  $P[u_v] \leftarrow P[u'_v] \leftarrow u$;
  $P[v_u] \leftarrow P[v'_u] \leftarrow v$;
  $E' \leftarrow E' \cup \{(u_v, v'_u), (v_u, u'_v)\}$;
  $c'((u_v, v'_u)) \leftarrow c'((v_u, u'_v)) \leftarrow \kappa * ||u - v||$;
  $I[u] \leftarrow I[u] \cup \{u_v, u'_v\}$;
  $I[v] \leftarrow I[v] \cup \{v_u, v'_u\}$;
  $A[u_v] \leftarrow \angle(u - v)$;
  $A[u'_v] \leftarrow \angle(v - u)$;
  $A[v'_u] \leftarrow \angle(u - v)$;
  $A[v_u] \leftarrow \angle(v - u)$;
`/* Add start and goal configuration */`
$V' \leftarrow \{s', g'\}$;
$I[s] \leftarrow I[s] \cup \{s'\}$;
$I[g] \leftarrow I[g] \cup \{g'\}$;
$A[s'] = \alpha$;
$A[g'] = \beta$;
`/* Add internal 'rotation arcs' for every point */`
for $v \in V$ **do**
  $v_0, \ldots, v_m \leftarrow$ sort $I[v]$ s.t. $\forall i = 0, \ldots m-1 : A[v_i] \leq A[v_{i+1}]$;
  for $i \in 0 \ldots m-1$ **do**
    $E' \leftarrow E' \cup \{(v_i, v_{i+1}), (v_{i+1}, v_i)\}$;
    $c'((v_i, v_{i+1})) \leftarrow c'((v_{i+1}, v_{i.})) \leftarrow \tau * (A[v_{i+1}] - A[v_i])$;
  $E' \leftarrow E' \cup \{(v_m, v_0), (v_0, v_m)\}$;
  $c'((v_m, v_0)) \leftarrow c'((v_0, v_m)) \leftarrow \tau * (2 * \pi + A[v_0] - A[v_m])$;
`/* Use the new graph to calculate the minimum cost path */`
Calculate shortest path p from s' to g' in $D(V', E')$ with cost function c';
Remove p_i from $p = (p_0, \ldots, p_m)$ if $P[p_{i-1}] = P[p_i]$;
return $(P[p_0], \ldots, P[p_m])$ for $p = (p_0, \ldots, p_m)$;

between the two loops, only a constant time is needed and only a constant number of vertices is added to G'. As m is in $O(|V(\mathcal{O})|)$, the second loop has a complexity of $O(n^2 + n * n * \log n) = O(n^2 * \log(n))$. In it only edges $(O(n^2))$ but no new vertices are added to G'. The final graph G' has $|V'| \in O(n^2)$ and $|E'| \in O(n^2)$. Hence, the shortest path can be calculated in $O(n^2 + n^2 * \log(n^2)) = O(n^2 \log n)$ by using Dijkstra's algorithm with Fibonacci heap. The removal of redundant points can be done in $O(n)$. This results in an overall complexity of $O(n^2 * \log n)$. □

5.3 Approximation algorithms

Using the approximation technique proposed in Sec. 4.3.1, we are also able to provide approximation algorithms for the geometric variant. The technique allows us to use any edge weights as long as skipping an intermediate point does not increase the distance. This is obviously fulfilled in the geometric variant. We can easily calculate a cycle cover for full coverage as in Sec. 4.3.3 and also for penalty coverage by using an artificial additional orientation as in Sec. 4.3.4. By doubling a minimum spanning tree or a prize-collecting Steiner tree we can then obtain a tour as in Theorem 22 resp. Theorem 27. In the case that we only allow one orientation for passing a point, the approximation factor for full coverage can be improved since we can calculate an optimal cycle cover via minimum perfect matching. This section does not present any new techniques but rather shows that the corresponding techniques of the previous chapter are also applicable to the geometric variant.

Note that for all the proposed approximation algorithms, the approximation factor and the runtime both depend linearly on the maximum number of orientations ω which is assumed to be constant. Depending on how the orientations are implemented, the runtime can actually be exponential on the size (number of bits) of ω.

5.3.1 Full coverage

An approximation algorithm for the full coverage geometric cycle cover can be obtained by simply applying the Match-Max-Pairs-Algorithm (Alg. 1) with edge weights calculated by Alg. 3.

Theorem 32. *Let ω be the maximum number of orientations a point is allowed to be passed (i.e., $\forall p \in P : |\delta(p)| \leq \omega$), then there is a $2 * \omega$ approximation algorithm for a full coverage cycle cover. In case $\omega = 1$, the optimal solution can be obtained in polynomial time.*

Proof. Use the Match-Max-Pairs-Algorithm of Sec. 4.3.1. Every orientation has a vertex pair that represent the configuration of being at the specific point and heading into the respective two directions. The cost of transition between any two configurations can be optimally calculated in polynomial time as we discussed in Sec. 5.2. Since we have only ω vertex pairs per point, by Theorem 11 we have a $2 * \omega$ approximation. In the case of $\omega = 1$, the problem equals the minimum weight perfect matching problem. □

To connect the cycles provided by Theorem 32, we use a simple minimum spanning

tree. Doubling the edges results in cycles with u-turns on the original cycles which can be connected with no additional costs (but the u-turns from the doubling involve an extra cost).

Lemma 8. *Two cycles that intersect on a* 180° *turn of one of the cycles can be merged to a single cycle without increasing the turn and length costs.*

Proof. The correctness of this lemma can be easily seen in Fig. 5.5 where we replace the 180° turn by two turns whose sum is 180°. There is no problem with the intersection

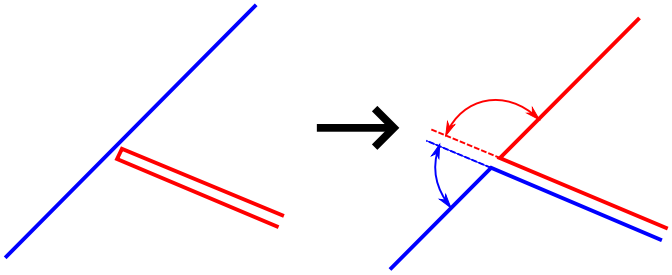


Figure 5.5: A u-turn that lies on another cycle allows a merging of the two cycle without additional costs. This also works if the intersection is on a vertex by moving it slightly onto a segment.

point also being a turn for the other cycle as can easily be seen by moving the intersection point by an infinitesimal ϵ such that it no longer does. Afterwards, we can straighten the part again, and hence remove the infinitesimal segment and its infinitesimal cost. □

Because the minimum spanning tree can be calculated optimally in polynomial time and is a lower bound on the tour, we can give constant factor bounds. The costs of the u-turns are charged to the cycles of the cycle cover.

Theorem 33. *Let* ω *be the maximum number of orientations a point is allowed to be passed, then there is a* $4 * \omega + 2$ *approximation algorithm for a full coverage tour. In case of* $\omega = 1$*, there is a* 4*-approximation algorithm.*

Proof. As in Theorem 22, we calculate a minimum spanning tree on the cycle cover approximation (Theorem 32) where the cost of an edge between two cycles equals the minimum path between any two vertices of the two cycles neglecting the heading at the two vertices. The minimum spanning tree is obviously a lower bound on the tour. For m cycles, there are $m - 1$ edges in the minimum spanning tree for which we obtain $m - 1$ additional cycles by doubling them. The additional costs are $2 * \mathrm{OPT} + (m - 1) * 2 * 180^\circ$ and we can merge all cycles to a tour by Lemma 8. As we know that the sum of turning angles in the m cycles of the cycle cover approximation already has to be at least $m * 360^\circ$ we can conclude that the obtained tour has a cost of at most $4 * \omega + 2$ times the optimum. In case that $\omega = 1$, we have an optimal cycle cover whose cost is obviously a lower bound for the optimum. □

5.3.2 Penalty coverage

Using the same methods as in Sec. 4.3.4, we can obtain similar results for the penalty coverage problems. First, for the cycle cover by adding an artificial orientation:

Theorem 34. *Let ω be the maximum number of orientations a point is allowed to be passed, then there is a $2 * (\omega + 1)$ approximation algorithm for a penalty cycle cover.*

Proof. As in Theorem 32 but we add an additional vertex pair with an edge that equals the penalty (in analogy to Theorem 25). □

From this penalty cycle cover we use the prize-collecting Steiner tree to select and connect good cycles. The connecting of the selected cycles via the edges of the tree is identical to full coverage. Only the analysis is slightly more difficult.

Theorem 35. *Let ω be the maximum number of orientations a point is allowed to be passed, then there is a $4 * (\omega + 1) + 4$ approximation algorithm for a penalty tour.*

Proof. This proof is analog to the proof of Theorem 27: We first calculate a penalty cycle cover approximation with a factor of $2 * (\omega + 1)$ using Theorem 34. We remove all points for which the penalty in the cycle cover has been paid. Next we calculate a 2-approximation of the prize-collecting Steiner tree using the approximation algorithm of Goemans and Williamson [53] that has a time complexity of $O(n^2 \log n)$. This is done on a graph that contains all remaining points and the edge costs equal the shortest path with turn costs but arbitrary start and goal headings. For two points that are in the same cycle, we set the costs to zero. We remove all cycles for which no point is in the resulting prize-collecting Steiner tree. All the other components we connect by selecting edges from the tree ($m - 1$ edges for m cycles). This can simply be done by iterating over all edges in the tree and adding it if it connects two different components. Obviously, the cost of an optimal prize-collecting Steiner tree is a lower bound for the tour (we only decreased edge costs). Due to the 2-approximation, the sum of all edge weights and penalties is at most 2 times the costs of the optimal penalty tour. The selected edges are transformed to cycles by doubling them and adding $180°$ turns at the ends. We can merge the cycles with no additional costs as in Theorem 33 using Lemma 8. This results in at most four times the cost of the optimal tour plus $2 * (m - 1) \times 180°$ turns for m cycles in the cycle cover. As every cycle in the cycle cover has also at least $360°$, we can charge the $180°$ turns onto the cycles which leads to $2 * 2 * (\omega + 1) * \mathrm{OPT}$. Combined, this results in $2 * 2 * (\omega + 1) * \mathrm{OPT} + 2 * 2 * \mathrm{OPT} = (4 * (\omega + 1) + 2) * \mathrm{OPT}$. All the steps are obviously in polynomial time for fixed ω. □

5.4 Integer programming

The same way as we generalized the approximation algorithms for grid graph instances (Sec. 4.3), we can generalize the second integer formulation in Sec. 4.4 for full coverage and penalty coverage. The only significant difference is how the edge weights are calculated. In the following, we provide the rough details for full coverage and leave the straightforward adaption for penalty coverage to the reader.

We work on an auxiliary graph $G(V, E)$: For every point $p \in P$ with the orientations $\delta(p)$, we create the vertices $V(p) = \bigcup_{\alpha \in \delta(p)} \{v_{p,\alpha}, v_{p,\alpha+\pi}\}$ representing the two configurations a point can be left/entered through one of its orientations. Further, there is an edge $e = \{v, v'\}$ between any two $v = v_{p,\alpha} \in V(p)$ and $v' = v_{p',\alpha'} \in V(p')$ with the cost $c(e)$ representing the minimum cost path from the configuration of being at p and heading α to the configuration of being at p' and heading opposite α' (i.e., $\alpha' + \pi$).

For the cycle cover variant, there is also an edge $e = \{v_{p,\alpha}, v_{p,\alpha+\pi}\}$ for every $p \in P$ with the cost of the cheapest cycle (covering at least two different points) using this passing for the cheapest such $\alpha \in \delta(p)$. An optimal tour would never use these edges but in a cycle cover, points can be in two unconnected cycles even in an optimal solution. The additional edge is used to implicitly represent this kind of cycles.

The integer programming formulation for cycle cover can be given as follows:

$$\min \quad \sum_{e \in E} c(e) * x_e \tag{5.1}$$

$$\text{s.t.} \quad \sum_{e \in E(v_{p,\alpha})} x_e = \sum_{e \in E(v_{p,\alpha+\pi})} x_e \quad \forall p \in P \forall \alpha \in \delta(v) \tag{5.2}$$

$$\sum_{\alpha \in \delta(p)} \sum_{e \in E(v_{p,\alpha})} x_e = 1 \qquad \forall p \in P \tag{5.3}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E \tag{5.4}$$

$$\tag{5.5}$$

Eq. 5.2 states that if and only if there is an incoming edge on one side, there has to be an outgoing edge on the opposite site. Eq. 5.3 states that there have to be exactly two edges entering/leaving (using the symmetry induced by the previous equation). $E(v)$ represents the set of edges incident to v.

The subcycle elimination constraints for obtaining a tour can also be adapted straightforward with only slight changes in the notation.

$$\sum_{e \in E(V(C), V(P \setminus C))} x_e \geq 2 \quad \forall C \subsetneq P, C \neq \emptyset \tag{5.6}$$

Hence, the integer programming formulation for tours is given by adding Eq. 5.6 to the cycle cover formulation.

5.5 Experiments

For the full coverage problem variants, we implemented the integer programs and the approximation algorithms for tour and cycle cover without obstacles. In this section we discuss the experimental results for these implementations. We compare the performance (time and objective value) for the integer programming approach for optimal solutions and the approximation algorithms. Further, we compare the performance for different resolutions ($\omega = 2, 3, 4$) to compare the influence on the runtime and the objective value. As cost weights we use $\tau = 1, \kappa = 0$ except for one additional run with $\omega = 2, \tau = 1, \kappa = 0.5$. The experiments were run for 10 random instances in the unit square per size

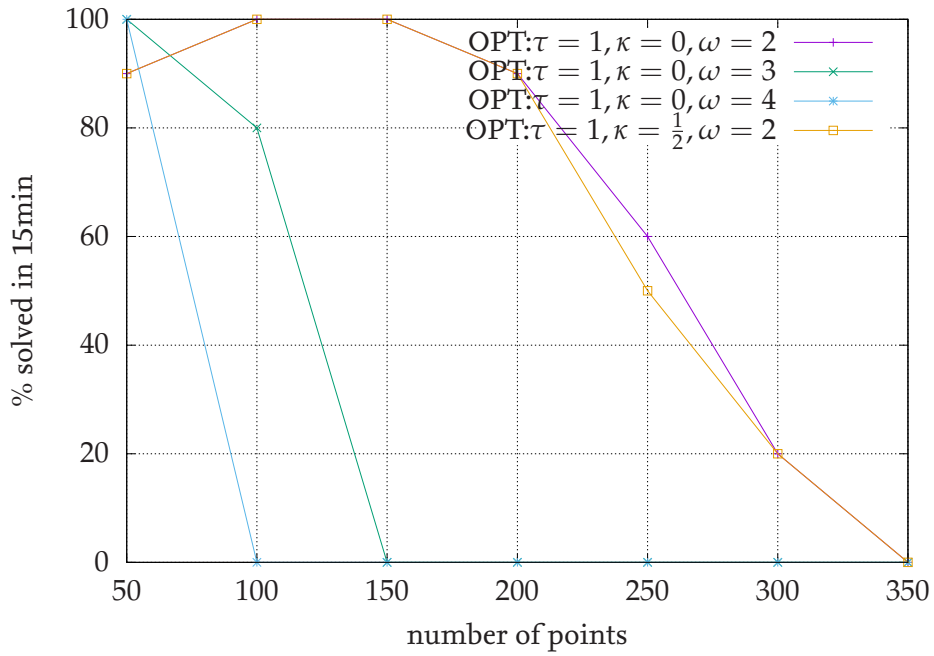


Figure 5.6: Percentage of cycle cover instances solved to optimum within 15 min. 10 instances for each size $50, 100, \ldots, 350$.

$50, 100, \ldots, 350$. As passing orientations $\{i * \frac{\pi}{\omega} | i = 0, \ldots \omega - 1\}$ were chosen for all points equally.

In Fig. 5.6 resp. Fig 5.7, the percentage of instances solved to optimum within 15 min via the integer program for cycle cover resp. tour is visualized. It can be seen that we can solve 50% of the instance of size up to 250 points resp. 200 points for $\omega = 2$. Already for $\omega = 3$ the performance drops strongly such that the maximum instance size becomes 100. For $\omega = 4$ only the smallest instances (50 points) have been solved in time. Hence, with our formulation only for $\omega = 2$ there is a serious advantage compared with the angular metric cycle cover and traveling salesman problem (using the work of Aichholzer et al. [9] as reference point). Interestingly, we can solve the tour version of an instance with 150 points for $\omega = 2$ in time but not the cycle cover version.

The average runtime is shown in Fig. 5.8 resp. Fig. 5.9. Note that cutting off the runtime after 15 min distorts the plot because the real runtime might be much higher. This leads to the sigmoidal look of the plots. The average runtime of the approximation algorithm is shown in Fig. 5.10. Here we do not differ between cycle cover and tour because they have nearly the same runtime. It can be seen that ω also has a lot of influence on the runtime of the approximation algorithm and the runtime for $\omega = 4$ grows significantly stronger than for $\omega = 2$. However, the runtime is much shorter than for the integer programs. The run with $\omega = 2, \tau = 1, \kappa = 0.5$ is nearly identical to the run with $\omega = 2, \tau = 1, \kappa = 0$

The objective value of the approximation algorithm differs for cycle cover only slightly

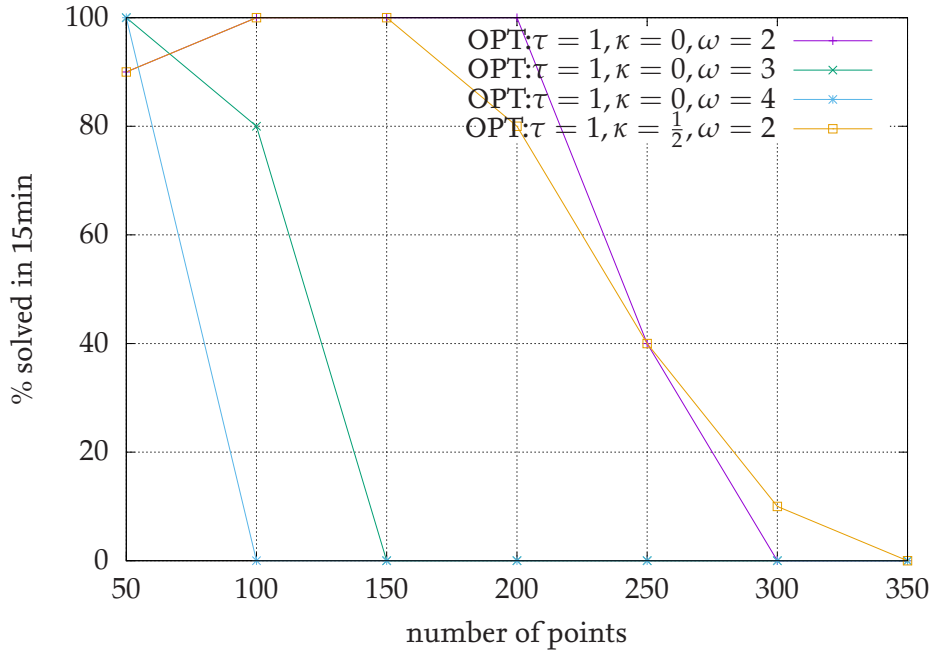


Figure 5.7: Percentage of tour instances solved to optimum within 15 min. 10 instances for each size $50, 100, \ldots, 350$.

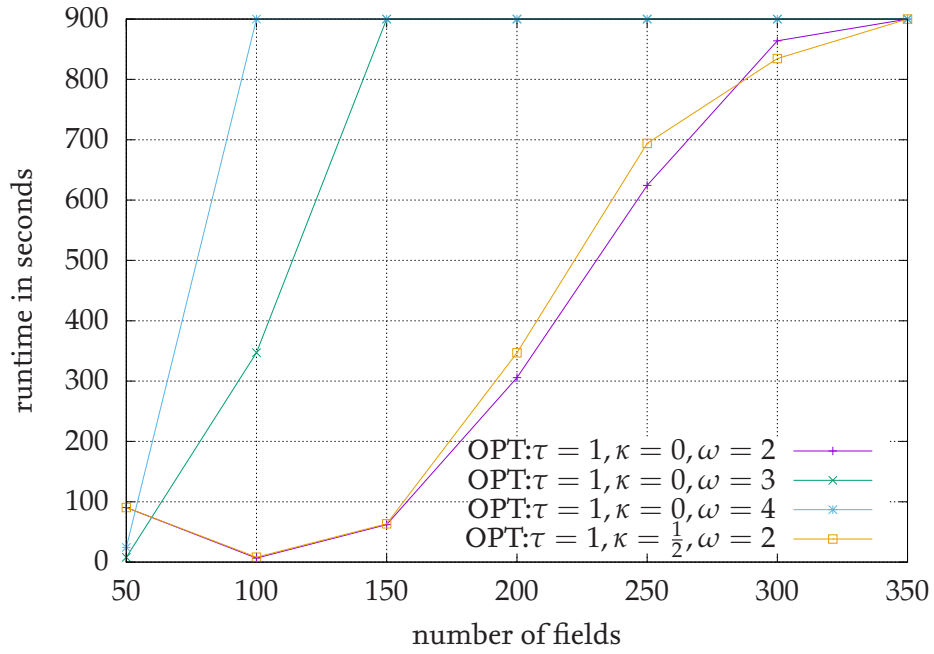


Figure 5.8: Average runtime of solving cycle cover instances to optimum with a cutoff after 15 min runtime. 10 instances for each size $50, 100, \ldots, 350$.

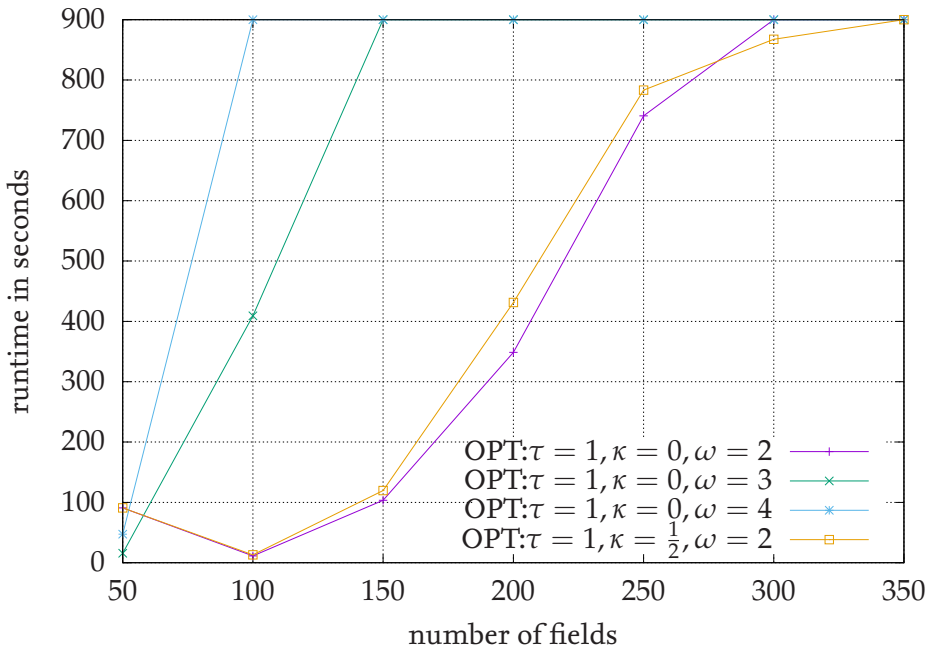


Figure 5.9: Average runtime of solving tour instances to optimum with a cutoff after 15 min runtime. 10 instances for each size $50, 100, \ldots, 350$.

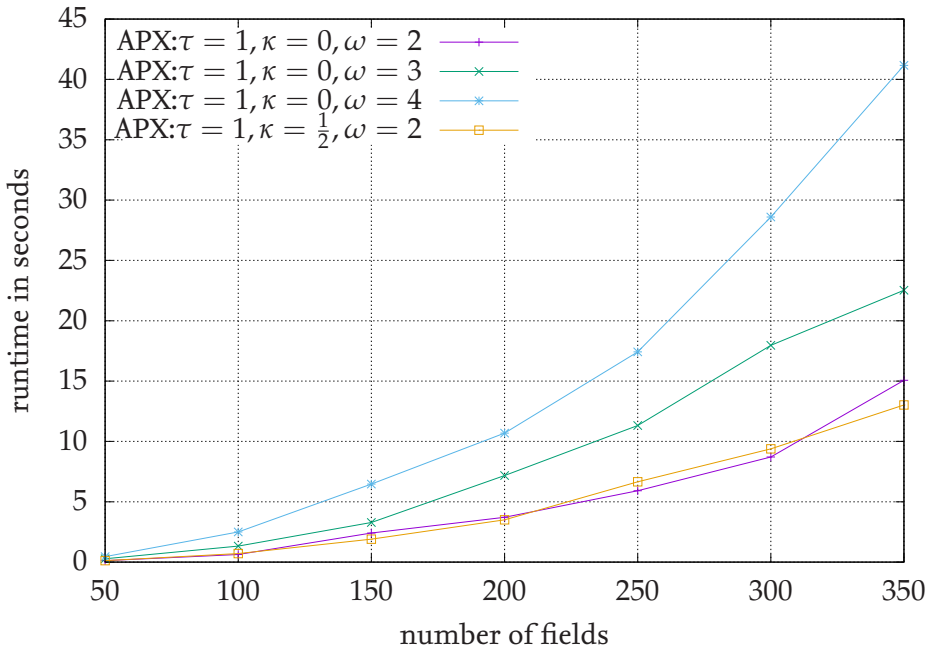


Figure 5.10: Average runtime of approximation algorithm for cycle cover and tour (nearly identical runtime). 10 instances for each size $50, 100, \ldots, 350$.

Size	$\omega = 2, \tau = 1, \kappa = 0$	$\omega = 3, \tau = 1, \kappa = 0$	$\omega = 4, \tau = 1, \kappa = 0$	$\omega = 2, \tau = 1, \kappa = 0.5$
50	1.031/9/1.067	1.073/10/1.178	1.088/10/1.129	1.028/9/1.069
100	1.024/10/1.04	1.076/8/1.106	-/-/-	1.034/10/1.057
150	1.044/10/1.073	-/-/-	-/-/-	1.037/10/1.057
200	1.036/9/1.057	-/-/-	-/-/-	1.034/9/1.049
250	1.039/6/1.046	-/-/-	-/-/-	1.038/5/1.049
300	1.033/2/1.036	-/-/-	-/-/-	1.028/2/1.03
350	-/-/-	-/-/-	-/-/-	-/-/-

Table 5.1: Difference of optimal and approximated solution for geometric cycle cover. The entries consist of *average difference/number of compared instances/maximal difference*. Only the instances that we were able to solve to optimum within 15 min are considered.

Size	$\omega = 2, \tau = 1, \kappa = 0$	$\omega = 3, \tau = 1, \kappa = 0$	$\omega = 4, \tau = 1, \kappa = 0$	$\omega = 2, \tau = 1, \kappa = 0.5$
50	1.09/9/1.368	1.173/10/1.304	1.18/10/1.484	1.08/9/1.225
100	1.089/10/1.242	1.227/8/1.457	-/-/-	1.149/10/1.325
150	1.152/10/1.308	-/-/-	-/-/-	1.12/10/1.247
200	1.15/10/1.268	-/-/-	-/-/-	1.096/8/1.164
250	1.103/4/1.141	-/-/-	-/-/-	1.097/4/1.109
300	-/-/-	-/-/-	-/-/-	1.053/1/1.053
350	-/-/-	-/-/-	-/-/-	-/-/-

Table 5.2: Difference of optimal and approximated solution for geometric tour. The entries consist of *average difference/number of compared instances/maximal difference*. Only the instances that we were able to solve to optimum within 15 min are considered.

from the optimum as can be seen in Table 5.1. For $\omega = 2$ it is on average less than 5% with a maximum difference of 7.3%. The differences are higher for $\omega = 3$ and $\omega = 4$ with a maximum difference of 17.8%, but there are too few instances solved for a reliable statement. The difference is higher for the tour version (Table 5.2), but still close to the optimum and far better than the proven bounds. The maximum difference found is 1.485 times the optimal value for $\omega = 4$. It has to be noted that the implementation does not do a local optimization of the cycle connections, i.e., the points of the connection are visited multiple times. A removal of the redundant visits could improve the ratio further.

The influence of the resolution on the objective value of the approximation algorithm can be seen in Table 5.3. The improvement from $\omega = 2$ to $\omega = 4$ is a few percents, and hence actually appreciable, but one also has to consider the increased runtime. Interestingly, for some instances the objective value with $\omega = 4$ is worse than with $\omega = 2$ (all orientations of $\omega = 2$ are also available for $\omega = 4$). We do not compare the optimal values because the solvable instance sizes for $\omega = 3$ and $\omega = 4$ are too small. The objective value represents only the objective value of the discretized problems and not of the corresponding geometric solution which might be better.

Size	Cycle Cover			Tour		
	$\omega = 2$	$\omega = 2$	$\omega = 4$	$\omega = 2$	$\omega = 3$	$\omega = 4$
50	37.519	35.837	34.135	40.038	39.24	37.02
100	60.612	56.816	55.186	64.52	64.497	59.734
150	81.389	76.941	72.578	89.863	82.912	79.471
200	97.589	93.206	88.585	108.318	101.939	92.817
250	114.237	109.049	105.434	122.456	116.373	111.404
300	131.013	125.596	120.601	136.978	132.809	125.984
350	143.887	139.266	134.495	153.869	147.526	141.928

Table 5.3: The average objective values for different resolutions ($\omega = 2, 3, 4$) using the approximation algorithm. The cost weights are $\kappa = 0, \tau = 1$.

Also larger instances have been solved with the approximation algorithm and the runtime can be seen in Fig. 5.11. We considered only random instances with $\omega = 2$, $\tau = 1$, and $\kappa = 0$ up to a size of 2000 points, but only present the results with up to 1700 points. For instances with 1700 points and more, the memory consumption of the linear program becomes a problem. One can see in the plot that the runtime roughly follows a curve but there are also some outliers. The reason for this could be that in some cases CPLEX has problems to solve the minimum weight perfect matching. Using a proper algorithm for it might solve this problem. However, the runtime of the minimum weight perfect matching has not been explicitly measured (only for some instances), and hence there might also be further bottlenecks.

The experiments were executed on modern 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 12.5.0.0 with the parameters `EpInt`=0, `EpGap`=0, `EpOpt`=1×10^{-9}, and `EpAGap`=0. No further optimizations have been performed.

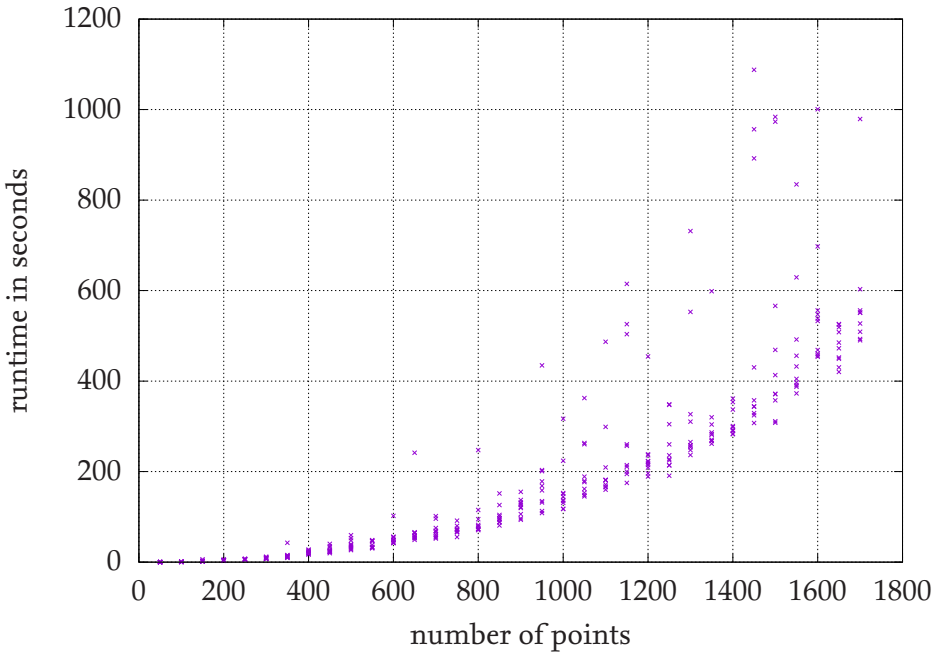


Figure 5.11: Runtime of approximation algorithm for instances of size $50, 100, \ldots, 1700$ with $\omega = 2$, $\tau = 1$, $\kappa = 0$ and 10 random instances per size. For instances of size 1700, the memory consumption becomes a problem. One instance for 1700 has been aborted for this reason.

6 Practical approach

In this short chapter, we propose heuristic approaches for planning tours for the mosquito drone. Instead of a clean model or theoretical correctness, we focus purely on practicality. This is contrary to the previous chapters of this thesis which contribute the primary results. Hence, this chapter can be considered as supplementary content that adds a different view on the problem. The two proposed heuristics have been quickly implemented and some rough and undocumented experiments have performed, but due to time constraints and an emphasis on the previous two chapters, a serious validation of these approaches is left to future work.

In Chapter 4, we discretized the environment into a grid and searched a tour within this grid. The expected amount of caught mosquitoes was modeled as the sum of the values of all visited fields (penalty coverage). As the drone can measure the collected mosquitoes in real time, the expected density for each field can be updated after each run. However, not only was even the polynomial time approximation algorithm not practical for reasonable instance sizes of, e.g., 1 km^2 with a field size of 1 m^2 but it also does not consider the limited battery. One can use a different approach and use larger fields that are only fully covered by a cycling movement if mosquitoes are caught in it. This approach leads to 'mosquito swarm search tours' with dynamic covering instead of 'direct covering tours' and could actually be useful in practice considering the limited amount of data. Especially, the correlation between the fields (if nothing is caught in a field, probably also the neighbored fields are empty) is better modeled with this approach. For dealing with the limited battery, one can scale down the penalties if the resulting tour is too expensive. Still, the previous methods are too expensive. In Chapter 5, we have individual points in a polygonal environment which would allow assigning a point to every hotspot and tour only on the hotspots. But these hotspots may be difficult to estimate and they are probably (large) areas and not points. Furthermore, any dynamic of the mosquito swarms is still ignored by these models. It shows that building a clean and reliable model for a practical tour optimization is rather difficult. Hence, it is not reasonable to rely on any such naïve model too much and try too hard to find a nearly optimal solution in this model. It is likely that at an early point, the tour does no longer improve in practice but only exploits weaknesses in the model.

Despite the focus on the mosquito hunting drone, the heuristics can, e.g., also be considered for cleaning robots that are not able to perform a full sweep. One can also use them for obtaining heuristic solutions for the grid graph problems by calculating the shortest path within the grid graph instead of the euclidean shortest path and sample vertices of the grid graphs instead of random coordinates.

6.1 Model and implementation

We now consider the problem model, the heuristics to optimize this model, and corresponding implementation details. The main part of the model is the density map (Sec. 6.1.1) that splits the area into a square grid and saves for every field the previous successes. The heuristics are based on sampling random points (Sec. 6.1.2) and trying to integrate them into the tour. To increase the average quality of these sampling points, the probability of them is dependent on the density map. We propose two heuristics, *Best Sampled Neighbor* (Sec. 6.1.3) and *Best Sampled Insertion* (Sec. 6.1.4, that are inspired by heuristics of the traveling salesman problem. Even in our 'quick and dirty'-implementation, both heuristics computed in first experiments on a notebook[1] (tens of) thousands of samples per second for 1000×1000 fields and a few simple obstacles. In Sec. 6.1.5, we propose some further (higher level) optimizations that build upon the two heuristics.

6.1.1 Density map

We need a simple model to somehow predict the mosquito densities for rating a possible tour. It should not only be efficient but also be adequate for the limited data we can collect. Furthermore, the model should also work with no data at all and learn by itself. The primary data we have is the positions of mosquito hits of previous runs.

We focus on 2-dimensional environments and put a simple square grid onto it but this would also work for 3-dimensional environments with a cube grid. Every field has a float value that represents the expected density which can also be seen in Fig. 6.1. The field can have a size larger than the drone especially if we want a tour that finds swarms and use a different technique as soon as we have found one. Obstacles are saved as additional polygons, but it can be useful to set unreachable fields to a neutral (zero) value.

After each run, the values of the visited fields are increased or decreased depending on if they were better or worse than expected. It can also be recommendable to also increase and decrease the neighbored fields slightly. If we have no information at the beginning, all fields can start with the same value and then get more accurate after each run. On this way, the model can learn on its own and adapt to changes.

We rate a tour based on the values of all visited fields but every field is counted at most once. Otherwise, most heuristics would probably only cycle over a small good area. This can be implemented efficiently by representing the density map as a 2-dimensional array where every field has two values: the density value and a counter for the visitations. If we add a new part to the tour, the counter of all newly visited fields is incremented and if the counter is zero for a field, its value is added to the overall sum. If we remove a part of the tour, the counter of all previously visited field is decremented and if the counter becomes zero again for a field, its value is removed from the overall sum.

Because many possible tour parts have to be considered for an optimization, the efficiency of querying the visited fields has to be high. Querying the visited fields, e.g. for a segment of the tour, in a classical multi-dimensional array is not optimal for the cache.

[1] *Intel(R) Core(TM) i5-5200U CPU* @ 2.20 GHz and 16 GB RAM

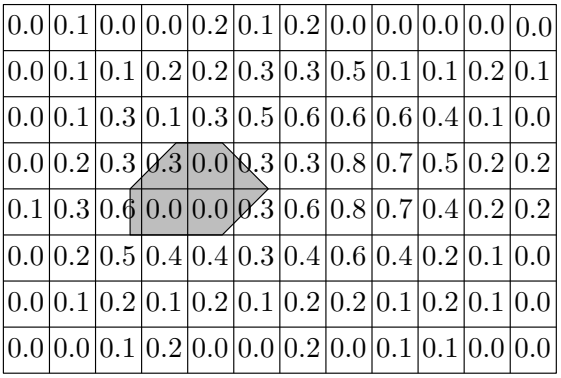


Figure 6.1: We use a square grid as a density map. It is updated after each run, i.e., a value in a field is decreased if the drone visited this field but was less successful than predicted and otherwise it is increased. Unvisited fields are not changed. The gray polygon represents an obstacle. In the beginning, all fields can have the same value if we have no information. The density map then will become more accurate after each run.

Only with a 50% chance is the neighbored field adjacent in the memory. However, the prefetcher can perform well since the necessary addresses can be independently calculated. Hence, the query efficiency should still be sufficient for most modern computers. First trials in improving the performance by using small submatrices have shown to be counterproductive.

Assuming neighbored fields to have similar values, the performance of querying a path (sequence of segments) can be increased by only considering, e.g., every second field. One can also not only consider the fields below the path but also the neighbored ones.

6.1.2 Density sensitive sampling

To increase the probability of a random sample point to be a good tour extension, we prefer to select points from areas that have a high density. More precise, we first select a field with a probability dependent on its density and in this field, we then select a random point. Of course, such a sampling process is more expensive than using a uniform distribution in especially since the densities change during the execution, i.e., become zero after the first visitation. In contrast, this sampling can be implemented more efficient than checking the quality of a sample point (at least for the assumed instances). Because the average sampling point quality can be expected to increase, we can obtain a higher quality with fewer sampling points, justifying the additional costs by the density sensitive sampling.

In our implementation, we first select a row based on the sum of fields in the row and then a field of this row. A row consist of fields that are adjacent in the memory, and hence can be efficiently read sequentially. The selection is performed by selecting a random value between zero and the sum of all corresponding rows/elements and iterate over the rows/elements until the sum of the visited rows/elements exceeds the random value. The element before this event is selected. The sum of fields for a row is saved in an array and

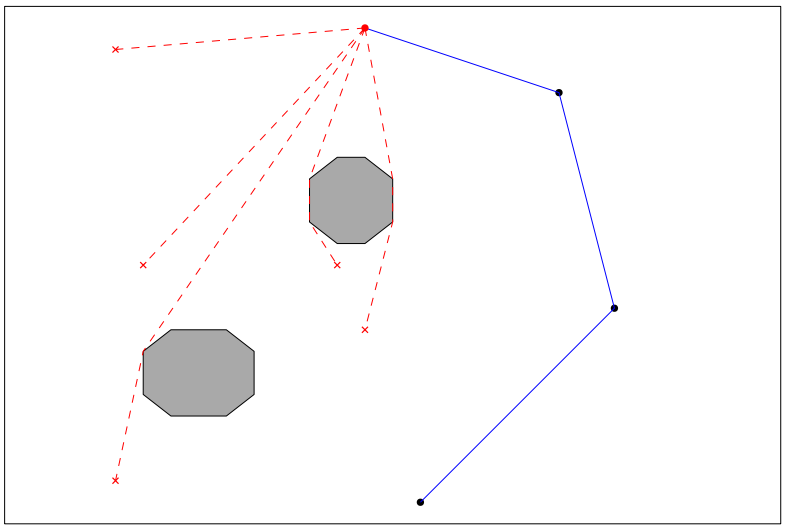


Figure 6.2: The Best Sampled Neighbor Heuristic incrementally builds a path (blue) by choosing from a set of random points (red) the one that has the best coverage/cost ratio.

automatically updated when a field becomes visited or unvisited. Since sequential reads in arrays can be performed fast and the corresponding arrays only have the length of the root of the number of fields, the sampling of points using this method is negligible compared to querying the quality of a path. Summing up a lot of small floating point numbers can create numerical issues which have to be taken care of. The sampled coordinate is checked not to lie within an obstacle by the *landmark point location* of *CGAL* [76].

6.1.3 Best Sampled Neighbor

A simple heuristic for the classical traveling salesman problem is to start at an arbitrary vertex and always go to the next unvisited neighbored vertex with the cheapest transition costs. Hence, this heuristic, also called *Nearest Neighbor*, iteratively builds a path and closes it when every vertex has been visited. This approach also allows a simple and fast sampling based heuristic for our coverage problem in the two-dimensional space with polygonal obstacles. In every iteration, we choose N random (feasible) positions and calculate a connection (e.g., euclidean shortest path) between every of these positions to the end position of the current path. We select the connection that has the highest coverage improvement per cost increase and add it to the current path. In order not to violate the bound on the maximal tour cost, we skip all of the N random points whose integration leads to higher costs than this bound. As soon as all of the N points have been skipped, we close the tour. To highlight that we use the vertex with the best bang-for-the-buck from a set of samples, we call this heuristic *Best Sampled Neighbor*. The idea is also shown in Fig. 6.2.

One of the main arguments for this heuristic is that we only have to do $2 * N$ euclidean shortest path calculations (from every random point to both ends of the tour) and a static coverage calculation of N paths. Static coverage calculation means that we do not change

the previous coverage, i.e., no fields become uncovered. For the heuristic in the next section, this is not the case. If the complexity of the obstacles is relatively low, the euclidean shortest path can be calculated fast in practice, especially if we precalculate and reuse the visibility graph of the obstacle vertices. This allows us a high sample rate, and hence a fast computation of a tour.

In Alg. 4 this method is given as pseudocode. It uses the following auxiliary functions: $T_{\mathcal{O}}(p_0, \ldots, p_n)$ inserts necessary turn points into a sequence of control points where between two subsequent control points p_i and p_{i+1}, the segment (p_i, p_{i+1}) might not necessarily be valid but could intersect some obstacle. The segments induced by $T_{\mathcal{O}}(p_0, \ldots, p_n)$, in contrast, are always feasible.

$$T_{\mathcal{O}}(p_0, \ldots, p_n) = \begin{cases} p_0, \mathrm{ESP}_{\mathcal{O}}(p_0, p_1), p_1, \mathrm{ESP}_{\mathcal{O}}(p_1, p_2), & \forall i = 1, \ldots, n : \mathrm{ESP}_{\mathcal{O}}(p_{i-1}, p_i) \neq \bot \\ \quad \ldots, p_{n-1}, \mathrm{ESP}_{\mathcal{O}}(p_{n-1}, p_n), p_n & \\ \bot & \text{else} \end{cases}$$

The inserted turn points are calculated by the euclidean shortest path $\mathrm{ESP}_{\mathcal{O}}(p_i, p_{i+1})$ between any two subsequent control points p_i and p_{u+1} around the obstacles $\mathcal{O}$. $\mathrm{ESP}_{\mathcal{O}}(p_i, p_{i+1})$ only returns the turn points and not the end points. We use the euclidean shortest path instead of the minimum cost path (as described in Sec. 5.2) because it is usually identical or very similar but faster to calculate. Note that for obtaining a valid tour from $T_{\mathcal{O}}(p_0, \ldots, p_n)$, p_0 and p_n have to be identical.

The cost of a sequence of points $p_0, \ldots, p_n$ that induces feasible segments is denoted by $\mathrm{cost}(p_0, \ldots, p_n)$ and contains the length costs of all segments as well as the turn costs between all subsequent segments. For simplicity, we ignore the turn cost at $p_0 = p_n$ for tours. The coverage of a sequence of points $p_0, \ldots, p_n$ that induces feasible segments is denoted by $\mathrm{cov}(p_0, \ldots, p_n)$. In our implementation, it is just a sum of a heuristically calculated coverage of its induced segments. In practice, one does of course not really recalculate $\mathrm{cost}(p_0, \ldots, p_n)$ or $\mathrm{cov}(p_0, \ldots, p_n)$ but exploits that we only insert a single point in each iteration. Hence, one can simply only calculate the relatively simple difference. For checking if the cost is still within the bounds, one can also use additional strategies before calculating the closing euclidean shortest path.

One thing to remark on this approach is that it also works online, hence we can adapt the density map at any time. Once a part of a tour has been scheduled, it is not changed again and also the point of time it appears in the tour is fixed. Because no field ever becomes uncovered again, the density map does not need a visitation counter but we can manipulate the field values directly. This also allows, e.g., to implement a dynamism as increasing the value of all covered fields every five minutes in which case also a double coverage after some time might be useful. Another possibility is to save the time of the last visit for each field and calculate dynamically calculate the value based on this. The heuristic in the next section does not allow this because it also changes previous parts of the tour.

Algorithm 4: Best Sampled Neighbor Heuristic (naïve implementation)

Data: p_{start}: start point, $c_{\max}$: maximal cost, $\mathcal{D}$: density map, $\mathcal{O}$: obstacle map
Result: Tour t with $\text{cost}(t) \leq c_{\max}$

$C \leftarrow (p_{\text{start}}, p_{\text{start}})$;
repeat
  $P' \leftarrow \perp$;
  for $\imath \in 1, \ldots, N$ **do**
    $p \leftarrow$ sample_point$(\mathcal{O}, \mathcal{D})$;
    $C'' \leftarrow C$ with p inserted before last position;
    $P'' \leftarrow C''$ excluding last element;
    $P \leftarrow C$ excluding last element;
    if $T_{\mathcal{O}}(C'') \neq \perp \wedge cost(T_{\mathcal{O}}(C'')) \leq c_{max}$ **then**
      if $P' = \perp \vee cov(T_{\mathcal{O}}(P''))/cost(T_{\mathcal{O}}(P'')) > cov(T_{\mathcal{O}}(P'))/cost(T_{\mathcal{O}}(P'))$ **then**
        $P' \leftarrow P''$;
  if $P' \neq \perp$ **then**
    $C \leftarrow P' + (P'[0])$;
    Update $\mathcal{D}$;
until $P' \neq \perp$;
return $T_{\mathcal{O}}(C)$;

6.1.4 Best Sampled Insertion

Another heuristic for the classical traveling salesman problem is to start with an initially incomplete tour (e.g., on the two vertices of the cheapest edge) and then iteratively extend this tour by choosing an uncovered vertex v and replacing an edge $\{u, w\}$ of the tour by $\{u, v\}$ and $\{v, w\}$ such that the cost increase is minimal. Since we have a feasible tour in every iteration, this is a natural candidate for tours with cost limits. If the vertex v is selected such that the cost increase is minimal, this heuristic is called *Cheapest Insertion*. In case it is selected such that the increase is maximal, the heuristic is called *Farthest Insertion*. In practice, Farthest Insertion often performs better for the traveling salesman problem than Cheapest Insertion because it tackles the expensive points in the beginning while the shape of the tour is less constrained. For our problem, a strategy like Farthest Insertion is probably no good idea because we do not need to visit expensive fields but want to visit as many good fields as possible in limited time.

In this section, we propose an approach similar to Cheapest Insertion and call it *Best Sampled Insertion*. In every iteration step, we have a tour and select N random points. For each of these random points, we check on which edge it can probably be integrated best. For this, we ignore obstacles and coverage as otherwise the cost for considering a single sample point would be too high. Only when the insertion point is chosen, we calculate/estimate the cost/coverage difference. Of all N possible extended tours, we choose

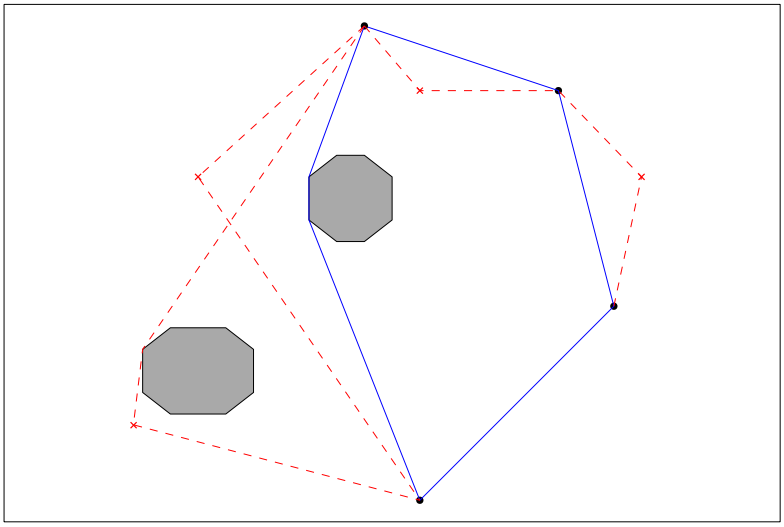


Figure 6.3: Best Sampled Insertion replaces a connection between two control points of the current tour (blue) by integrating a sample point (red) in between. For every sample point, the connection with the cheapest estimated insertion costs is selected. Vertices necessary to bypass obstacles are not control points and possibly removed by such an integration.

the one with the best coverage/cost ration. This approach is visualized in Fig. 6.3. Because we first have to find the position where we can insert the new point best and also remove an edge of the previous tour (hence, the coverage can decrease), this is computationally more expensive than the previous heuristic. However, we can implement this approach to still perform a relatively high sampling rate.

The pseudocode for the heuristic is given in Alg. 5. We use the same auxiliary methods as for *Sampled Best Neighbor*. Same as for *Sampled Best Neighbor*, they are only for simplicity and in practice, one would only consider the differences to the current tour after inserting the new point instead of recalculating everything. This means that cost() and cov() can be implemented much more efficient than shown in the pseudocode by only considering the relatively small difference to C.

Because this approach is constantly changing all parts of the tour, the density map has to be static and cannot be changed from outside during the execution.

6.1.5 Further optimization

The simplest improvement can be achieved by applying a local optimization on the produced tours. One can randomly select a control point of the tour and randomly shift it slightly or remove it and check if the tour improves.

Instead of using only a single sample point in each iteration, one can also think of using multiple of them. This can lead to better tours in general, but can also increase the computational complexity. The simplest case of this is using two sample points and integrate the path between these two points into the current solution. One might want to limit the distance between these two points or even only allow segments. For more points,

Algorithm 5: Best Sampled Insertion Heuristic (naïve implementation)

Data: p_{start}: start point, $c_{\max}$: maximal cost, $\mathcal{D}$: density map, $\mathcal{O}$: obstacle map
Result: Tour t with $\text{cost}(t) \leq c_{\max}$

$C \leftarrow (p_{\text{start}}, p_{\text{start}})$;
repeat
  $C' \leftarrow \perp$;
  for $\imath \in 1, \ldots, N$ **do**
    $p \leftarrow$ sample_point$(\mathcal{O}, \mathcal{D})$;
    Find i with $(|CP[i] - p| + |p - CP[i+1]|)/|CP[i+1] - CP[i]|$ minimal;
    $C'' \leftarrow C$ with p inserted between $C[i]$ and $C[i+1]$;
    if $T_{\mathcal{O}}(C'') \neq \perp \wedge cost(T_{\mathcal{O}}(C'')) \leq c_{max} \wedge cov(T_{\mathcal{O}}(C'')) \geq cov(T_{\mathcal{O}}(C))$ **then**
      if $C' = \perp \vee cov(T_{\mathcal{O}}(C''))/cost(T_{\mathcal{O}}(C'')) > cov(T_{\mathcal{O}}(C'))/cost(T_{\mathcal{O}}(C'))$ **then**
        $C' \leftarrow C''$;
  if $C' \neq \perp$ **then**
    $C \leftarrow C'$;
    Update $\mathcal{D}$;
until $C' \neq \perp$;
return $T_{\mathcal{O}}(C)$;

one has a hamiltonian path problem which becomes difficult to solve for larger sets. Of course, one can also simply use the basic heuristics but let them run some iterations, reset to the previous tour, repeat this a few times, and select the best new tour.

As the heuristics are fast enough to create a larger set of tours and are easy to merge, one can also think of using a genetic algorithm to further improve the quality. For this, we first create a pool of instances using the previous heuristics and then randomly select two instances (good instances with higher probability) and cross them. The crossing can, e.g., be done by finding the two closest points in them and exchange the end parts at this position. As we only change a relatively small part, the evaluation of the new tours can be done relatively fast. Also a mutation by randomly replacing a point by another point within a specific range can be easily implemented. For more details on genetic algorithms consider, e.g., the book by Talbi [77].

7 Conclusion

This thesis was structured in three main parts: grid graphs (Chapter 4), geometric instances (Chapter 5), and practical approaches (Chapter 6). The most extensive part was the chapter about grid graphs, where we considered different problem variants for tours and cycle covers with turn and length costs. The full coverage problem variant has been considered before while the subset coverage and the penalty coverage variants are new. We considered NP-hardness, approximation algorithms, and integer programming for them, obtained new results for the full coverage variant in all three areas, and were able to generalize these results to subset coverage and penalty coverage. The chapter on geometric instances builds upon the results of the previous chapter but carries them into a polygonal environment. While the passing orientations for a point are still restricted, the transitions between two points are now arbitrary geometric paths. These results are interesting for comparison with the research on the angular metric traveling salesman problem where the passing orientations are not restricted. The last chapter on practical approaches for the mosquito hunting drone roughly described more practical solutions but we only scratched the surface.

Grid graph We first showed in Sec. 4.2.2 that cycle cover with turn costs in 2-dimensional grid graphs is NP-hard which answers the open *Problem* 53 in *The Open Problems Project* [2] and is probably the most significant result of this thesis. The 'interesting fields' in this proof are in the center of a 5×5 subgraph such that the question arises if the problem is also hard for graphs that do not contain 3×3, 4×4, etc. subgraphs. For grid graphs that do not contain 2×2 subgraphs (thin grid graph), Arkin et al. [3] have already shown that the problem is solvable in polynomial time. One can also think about the hardness in simple grid graphs (which contain no holes). For subset cycle cover, we have shown in Sec. 4.2.3 that the problem is already NP-hard in thin grid graphs by adapting a proof by Aggarwal et al. [1]. Both hardness proofs do not give lower bounds on the achievable approximation factor.

In Sec. 4.3 we gave constant factor approximation algorithms for all considered problem variants and not only for 2-dimensional grid graphs but also for general grid graphs such as hexagonal grids or 3-dimensional grids. To the best of our knowledge, we gave the first approximation algorithms for the subset and the penalty coverage variant. They are a rather simple adaption of the technique we also use for the full coverage variant. For general grid graphs our proven approximation factor for full coverage is better than the method of Arkin et al. [3] but not for 2-dimensional grid graphs. Due to the expensive linear program we use, our approximation algorithm is also of relatively low practicality. Assuming a similar performance as the analog approximation algorithm for geometric instances (see Sec. 5.5), one can assume that without further engineering the solvable

instance size on a modern desktop computer (in 2016) is less than 2000 fields. For all problem variants, better approximation factors should be achievable. There is also an interest in improving the computational complexity or at least improve the engineering. The main problem is the large linear program (especially the high memory consumption of CPLEX) and there are techniques like column generation to ease this problem.

We considered a set of integer programming formulations for all problem variants in Sec. 4.4 and provided an experimental evaluation of them in Sec. 4.5. For full coverage tours we increased the solvable instance size by a significant factor compared to the results of de Assis and de Souza [4]. We noticed that the performance of the tour variants is mostly only slightly worse than the performance of the cycle cover variants and that our sparse instances are harder to solve. More experiments with more kinds of instances should be made to identify the difficult parts in the instances. By observing the CPLEX logs, one can also notice that often the solver obtains quickly a nearly optimal solution for the cycle cover problem. By tuning the parameters in CPLEX, one can possibly improve the performance. One could also imagine to optimize large instances by partitioning the area into small areas and solve the small areas to near optimum with CPLEX and then connect the solutions. There were some problems with some instances for subset and penalty coverage for some of the formulations which still need a more careful consideration. All experiments have been performed with only turn costs and especially the tour problems might perform differently with additional distance costs.

One problem that we did not consider but that often appears in practice is the additional constraint of a maximal cost (e.g. limited battery) for the penalty coverage version.

Geometric instances In Sec. 5.3, we generalized the approximation algorithms for general grid graphs to a more geometric problem variant. The corresponding full coverage problems can be considered as a discretized version of the angular metric traveling salesman problem/cycle cover with obstacles. First, we have seen in Sec. 5.2 that the euclidean shortest path including turn costs between two configurations can easily be calculated via the visibility graph similar to the traditional euclidean shortest path. The approximation factors for the full coverage and the penalty coverage approximation algorithms only depend on the resolution of the passing orientation limitation. They do not depend on the problem size like the $O(\log n)$ approximation algorithms for the angular metric TSP/cycle cover by Aggarwal et al. [1]. Simple IP-formulations for our discretized problem variant are discussed in Sec. 5.4. In Sec. 5.5, we provided an experimental evaluation for the integer programs and the approximation algorithm for the full coverage problem variant without obstacles. It showed, that for 2 orientations (horizontal and vertical), we were able to solve instances with up to 300 points. This is significantly more than Aichholzer et al. [9] were able to for the non-discretized version. Interestingly, already for 3 orientations, the performance dropped drastically. Our approximation algorithm was able to solve instances of size close to 2000 vertices for two orientations. The solutions were relatively close to the optimal solutions of the integer programs and far below the proven factor. As mentioned for grid graphs, the memory consumption of the linear program

is critical and can probably be improved by more careful engineering. By using a proper minimum weight perfect matching algorithm instead of an integer program one should be able to further improve the runtime and reduce outliers. There should still be a lot of room to improve the computational complexity and the approximation factors of the approximation algorithms.

Practical approaches We considered a simple optimization model for the mosquito hunting drone (or similar problems) and two corresponding simple heuristics. In a quick implementation and first undocumented experiments, they produced solutions even for grids with 1000×1000 fields within a few seconds. We only mentioned further optimization possibilities and also did not test the heuristics in realistic experiments. This has mostly been due to time constraints and the general focus of this thesis on the previous two subjects.

Bibliography

[1] Alok Aggarwal, Don Coppersmith, Sanjeev Khanna, Rajeev Motwani, and Baruch Schieber. The angular-metric traveling salesman problem. *SIAM Journal on Computing*, 29(3):697–711, 2000.

[2] Erik D. Demaine, Joseph S. B. Mitchell, and O'Rourke Joseph. The open problems project. `http://cs.smith.edu/~orourke/TOPP/`.

[3] Esther M Arkin, Michael A Bender, Erik D Demaine, Sándor P Fekete, Joseph SB Mitchell, and Saurabh Sethia. Optimal covering tours with turn costs. *SIAM Journal on Computing*, 35(3):531–566, 2005.

[4] Igor R De Assis and Cid C De Souza. Experimental evaluation of algorithms for the orthogonal milling problem with turn costs. In *Experimental Algorithms*, pages 304–314. Springer, 2011.

[5] Esther M Arkin, Sándor P Fekete, and Joseph SB Mitchell. Approximation algorithms for lawn mowing and milling. *Computational Geometry*, 17(1):25–50, 2000.

[6] Alon Itai, Christos H Papadimitriou, and Jayme Luiz Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982.

[7] Olaf Maurer. Winkelminimierung bei Überdeckungsproblemen in graphen. Diplomarbeit, Technische Universität Berlin, 2009.

[8] Cplex optimizer. `https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`.

[9] Oswin Aichholzer, Anja Fischer, Frank Fischer, J Fabian Meier, Ulrich Pferschy, Alexander Pilz, and Rostislav Stanek. Minimization and maximization versions of the quadratic traveling salesman problem. Preprint of the Institute for Numerical and Applied Mathematics of the University of Goettingen, March 2016.

[10] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

[11] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation - international edition (2. ed)*. Addison-Wesley, 2003.

[12] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2009.

[13] Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

[14] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. *Computational Geometry*. Springer, 2008.

[15] Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[16] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1982.

[17] Sartaj Sahni and Teofilo Gonzalez. P-complete approximation problems. *Journal of the ACM (JACM)*, 23(3):555–565, 1976.

[18] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, DTIC Document, 1976.

[19] George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410, 1954.

[20] D Applegate, R Bixby, V Chvtal, and W Cook. Concorde tsp solver. `http://www.math.uwaterloo.ca/tsp/concorde.html`.

[21] Joseph SB Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric tsp, k-mst, and related problems. *SIAM Journal on Computing*, 28(4):1298–1309, 1999.

[22] Sanjeev Arora. Polynomial time approximation schemes for euclidean tsp and other geometric problems. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, pages 2–11. IEEE, 1996.

[23] William Cook. *In pursuit of the traveling salesman: mathematics at the limits of computation*. Princeton University Press, 2012.

[24] David L Applegate, Robert E Bixby, Vasek Chvatal, and William J Cook. *The traveling salesman problem: a computational study*. Princeton university press, 2011.

[25] Esther M Arkin, Michael A Bender, Erik D Demaine, Sándor P Fekete, Joseph SB Mitchell, and Saurabh Sethia. Optimal covering tours with turn costs. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 138–147. Society for Industrial and Applied Mathematics, 2001.

[26] M Fellows, Panos Giannopoulos, Christian Knauer, Christophe Paul, F Rosamond, Sue Whitesides, and Nathan Yu. Abstract milling with turn costs. *arXiv preprint arXiv:0912.1050*, 2009.

[27] Nadia M. Benbernou. *Geometric Algorithms for Reconfigurable Structures*. phdthesis, Massachusetts Institute of Technology, 2011.

[28] Clifford Stein and David P Wagner. *Approximation algorithms for the minimum bends traveling salesman problem.* Springer, 2001.

[29] Simeon Ntafos. Watchman routes under limited visibility. *Computational Geometry*, 1(3):149–170, 1992.

[30] Kazuo Iwano, Prabhakar Raghavan, and Hisao Tamaki. *The traveling cameraman problem, with applications to automatic optical inspection.* Springer, 1994.

[31] Enric Galceran and Marc Carreras. A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, 61(12):1258–1276, 2013.

[32] Alexander Zelinsky, Ray A Jarvis, JC Byrne, and Shin'ichi Yuta. Planning paths of complete coverage of an unstructured environment by a mobile robot. In *Proceedings of international conference on advanced robotics*, volume 13, pages 533–538, 1993.

[33] Yoav Gabriely and Elon Rimon. Spiral-stc: An on-line coverage algorithm of grid environments by a mobile robot. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 1, pages 954–960. IEEE, 2002.

[34] Wesley H Huang. Optimal line-sweep-based decompositions for coverage algorithms. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 1, pages 27–32. IEEE, 2001.

[35] Zhiyang Yao, Satyandra K Gupta, and Dana S Nau. Hybrid cutter path generation for 2.5 d milling operation. In *ASME 2002 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 703–714. American Society of Mechanical Engineers, 2002.

[36] Michel Taïx, Philippe Souères, Helene Frayssinet, and Lionel Cordesses. Path planning for complete coverage with agricultural machines. In *Field and service robotics*, pages 549–558. Springer, 2003.

[37] Osman Ali, Bart Verlinden, and Dirk Van Oudheusden. Infield logistics planning for crop-harvesting operations. *Engineering Optimization*, 41(2):183–197, 2009.

[38] Ali Ahmadzadeh, James Keller, George Pappas, Ali Jadbabaie, and Vijay Kumar. An optimization-based approach to time-critical cooperative surveillance and coverage with uavs. In *Experimental Robotics*, pages 491–500. Springer, 2008.

[39] Amit Agarwal, Meng-Hiot Lim, Meng-Joo Er, and Chan Yee Chew. Aco for a new tsp in region coverage. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 1717–1722. IEEE, 2005.

[40] Gerold Jäger and Paul Molitor. Algorithms and experimental study for the traveling salesman problem of second order. In *Combinatorial Optimization and Applications*, pages 211–224. Springer, 2008.

[41] 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. *Discrete Applied Mathematics*, 166:97–114, 2014.

[42] A. Fischer. *A Polyhedral Study of Quadratic Traveling Salesman Problems*. PhD thesis, Chemnitz University of Technology, 2013.

[43] Anja Fischer and Christoph Helmberg. The symmetric quadratic traveling salesman problem. *Mathematical Programming*, 142(1-2):205–254, 2013.

[44] Borzou Rostami, Francesco Malucelli, Pietro Belotti, and Stefano Gualandi. Quadratic tsp: A lower bounding procedure and a column generation approach. In *Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on*, pages 377–384. IEEE, 2013.

[45] Sándor P Fekete and Gerhard J Woeginger. Angle-restricted tours in the plane. *Computational Geometry*, 8(4):195–218, 1997.

[46] Sylvain Lazard, John Reif, and Hongyan Wang. The complexity of the two dimensional curvatureconstrained shortest-path problem. In *Proceedings of the Third International Workshop on the Algorithmic Foundations of Robotics,(Houston, Texas, USA)*, pages 49–57. Citeseer, 1998.

[47] Lester E Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of mathematics*, 79(3):497–516, 1957.

[48] Pankaj K Agarwal, Therese Biedl, Sylvain Lazard, Steve Robbins, Subhash Suri, and Suehl Whitesides. Curvature-constrained shortest paths in a convex polygon. *SIAM journal on computing*, 31(6):1814–1851, 2002.

[49] Pankaj K Agarwal and Hongyan Wang. Approximation algorithms for curvature-constrained shortest paths. *SIAM Journal on Computing*, 30(6):1739–1772, 2001.

[50] Jean-Daniel Boissonnat and Sylvain Lazard. A polynomial-time algorithm for computing a shortest path of bounded curvature amidst moderate obstacles. In *Proceedings of the twelfth annual symposium on Computational geometry*, pages 242–251. ACM, 1996.

[51] Ryo Takei, Richard Tsai, Haochong Shen, and Yanina Landa. A practical path-planning algorithm for a simple car: a hamilton-jacobi approach. In *American Control Conference (ACC), 2010*, pages 6175–6180. IEEE, 2010.

[52] Jerome Le Ny, Eric Feron, and Emilio Frazzoli. On the dubins traveling salesman problem. *IEEE Trans. Automat. Contr.*, 57(1):265–270, 2012.

[53] Michel X Goemans and David P Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.

[54] Giorgio Ausiello, Vincenzo Bonifaci, Stefano Leonardi, and Alberto Marchetti-Spaccamela. Prize-collecting traveling salesman and related problems, 2007.

[55] Keld Helsgaun. Solving the equality generalized traveling salesman problem using the lin–kernighan–helsgaun algorithm. *Mathematical Programming Computation*, 7(3):269–287, 2015.

[56] Takao Asano, Tetsuo Asano, Leonidas Guibas, John Hershberger, and Hiroshi Imai. Visibility of disjoint polygons. *Algorithmica*, 1(1-4):49–63, 1986.

[57] Emo Welzl. Constructing the visibility graph for n-line segments in o (n 2) time. *Information Processing Letters*, 20(4):167–171, 1985.

[58] Subir Kumar Ghosh and David M Mount. An output-sensitive algorithm for computing visibility graphs. *SIAM Journal on Computing*, 20(5):888–910, 1991.

[59] John Hershberger and Subhash Suri. An optimal algorithm for euclidean shortest paths in the plane. *SIAM Journal on Computing*, 28(6):2215–2256, 1999.

[60] John Canny and John Reif. New lower bound techniques for robot motion planning problems. In *Foundations of Computer Science, 1987., 28th Annual Symposium on*, pages 49–60. IEEE, 1987.

[61] Esther M Arkin, Joseph SB Mitchell, and Christine D Piatko. Bicriteria shortest path problems in the plane. In *Proc. 3rd Canad. Conf. Comput. Geom*, pages 153–156. Citeseer, 1991.

[62] Wei-pang Chin and Simeon Ntafos. Optimum watchman routes. *Information Processing Letters*, 28(1):39–44, 1988.

[63] Svante Carlsson, Håkan Jonsson, and Bengt J Nilsson. Finding the shortest watchman route in a simple polygon. In *International Symposium on Algorithms and Computation*, pages 58–67. Springer, 1993.

[64] Joseph SB Mitchell. Approximating watchman routes. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 844–855. Society for Industrial and Applied Mathematics, 2013.

[65] Muhammed H Alsuwaiyel and DT Lee. Minimal link visibility paths inside a simple polygon. *Computational Geometry*, 3(1):1–25, 1993.

[66] Frank K Hwang, Dana S Richards, and Pawel Winter. *The Steiner tree problem*, volume 53. Elsevier, 1992.

[67] Baruch Awerbuch, Yossi Azar, Avrim Blum, and Santosh Vempala. New approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. *SIAM Journal on Computing*, 28(1):254–262, 1998.

[68] Thomas J Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 216–226. ACM, 1978.

[69] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311. ACM, 1984.

[70] Jack Edmonds. Maximum matching and a polyhedron with 0, l-vertices. *J. Res. Nat. Bur. Standards B*, 69(1965):125–130, 1965.

[71] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467, 1965.

[72] William Cook and Andre Rohe. Computing minimum-weight perfect matchings. *INFORMS journal on computing*, 11(2):138–148, 1999.

[73] W.J. Cook, W.H. Cunningham, W.R. Pulleyblank, and A. Schrijver. *Combinatorial Optimization*. Wiley-Interscience, 1997.

[74] Carl Hierholzer and Chr Wiener. Über die möglichkeit, einen linienzug ohne wiederholung und ohne unterbrechung zu umfahren. *Mathematische Annalen*, 6(1):30–32, 1873.

[75] Ulrich Pferschy and Rostislav Staněk. Generating subtour elimination constraints for the tsp from pure integer solutions. *Central European Journal of Operations Research*, pages 1–30, 2016.

[76] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 4.9 edition, 2016.

[77] El-Ghazali Talbi. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons, 2009.