EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science
Algorithms Research Group

Minimum Displacement Weighted Graph Embedding

Master Thesis

Sasha Gielis

Supervisor:
Dr. K.A.B. Verbeek

Assessment committee:
Dr. K.A.B. Verbeek
Dr. T.C. van Dijk
Dr. A. Arleo

Final version

Eindhoven, October 2024

Abstract

Flow maps visualize movements between locations on a geographical map using lines with width proportional to the amount of flow being moved. An effective flow map minimizes overlap among flow lines and does not obscure critical map features. Existing methods primarily focus on the thematic layer of the map, leveraging techniques as rerouting and bundling of flow lines. However, the integration of flow lines with a geographical map is limited by geographical features, often necessitating the distortion of the underlying geographical layer. While automatic methods have been proposed that construct flow maps by distorting the flow lines, there are few algorithmic results on automatic map deformation for flow map construction. Our research contributes to the relatively unexplored area of automatic distortion in flow map design.

In this thesis, we introduce the Minimum Displacement Graph Embedding (MDGE) problem, which asks to find new positions for a set of point obstacles with minimum displacement to allow for the construction of a planar thick embedding of a weighted graph. The thick edges in the output are required to be homotopic to the input edges. While the one-dimensional MDGE problem can be efficiently solved, the two-dimensional variant becomes NP-hard if we allow the thick edges to overlap. We model the MDGE problem as a constrained optimization problem and prove that only imposing constraints on the edges of a Delaunay triangulation yields a 1.998-approximation of the constraints. Through the addition of constraints on the orthogonal order of the points, we are able to propose a practical and efficient algorithm for the MDGE problem, combining a linear program for displacing the obstacles with a method for computing thick homotopic edges.

Contents

Chapter 1

Introduction

A flow map is a well-known type of thematic map that uses lines to visualize movements between geographical locations. These movements, known as flows, may represent various types of data, such as highway traffic, migration patterns, or economic trade. In addition to communicating connectivity information through lines, flow maps often convey quantitative data by depicting each flow line with a width proportional to the amount of flow being moved. Arrow heads may be used to indicate the direction of flow. As can be seen in Figure 1.1, simply depicting the flows using straight-line arrows may lead to visual clutter due to overlapping flow lines, severely deteriorating the readability of the map. Furthermore, some of the islands are barely visible, making it more difficult to recognize the underlying geographical map. Another disadvantage of straight flow lines is that they fail to communicate the route along which flow moves, which is sometimes desired in flow maps depicting trade or migration. Hence, the integration of flow lines with a geographical map requires more careful attention to ensure the clear depiction of flow lines and critical map features such as lakes, buildings, or geographic regions.

Effective flow maps minimize overlap among flow lines and do not obscure critical map features. A common strategy is to bundle flow lines together, which provides a compact view of the flows but often obscures the individual flow routes, making it more difficult to distinguish between different flows. Traditional thematic map construction focuses primarily on the thematic layer of the map, that is, the information being shown. However, in flow maps, the depiction of the flow lines is often limited by geographical features, as these may not always leave sufficient space for visualizing the flows. Therefore, the design of flow maps may also require distortion of the underlying geographical

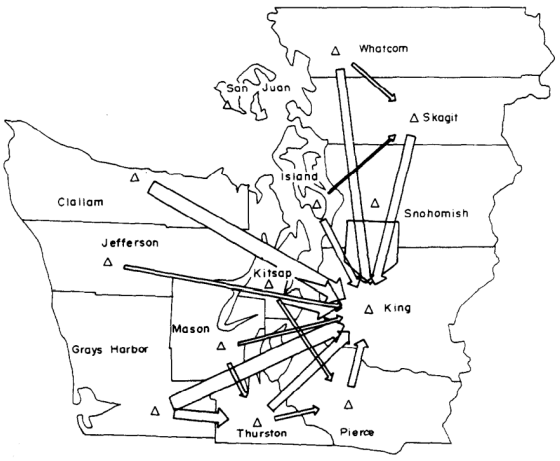


Figure 1.1: A flow map showing net referrals of cancer patients in Northwest Washington State between 1974 and 1978. Adapted from [28].

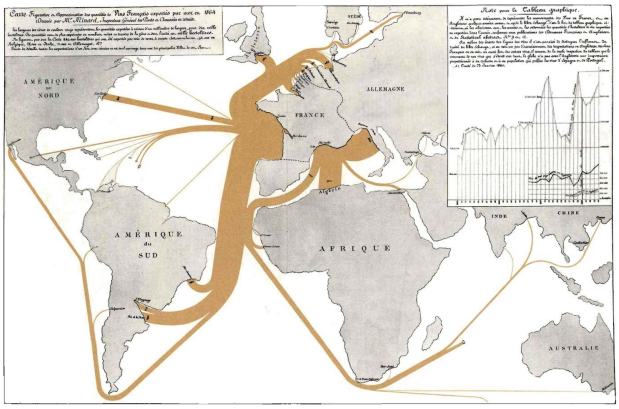


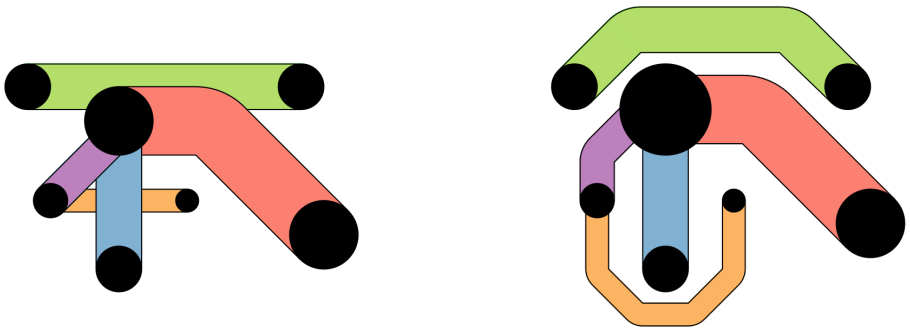
Figure 1.2: Minard's flow map showing wine exports from France in 1864. Adapted from [91].

layer to ensure the unobstructed integration of flows. At the same time, it is crucial for the readability and recognizability of flow maps to remain geographically accurate. This entails that the various map features maintain their relative positions with respect to the flow lines, and that their original positions, shapes and adjacencies are preserved as much as possible. Figure 1.2 shows an early hand-drawn flow map by Minard in which flow lines are bundled and geographic regions are distorted in favor of clear flow communication, while geographical accuracy is maintained. For instance, France is depicted larger to make room for the visualization of outgoing flows, but has retained its original position, characteristic shape and adjacencies with other countries. While automated methods have been proposed that address the discussed challenges by distorting the flow lines, there are few algorithmic results on automatic map deformation for flow map construction. As a result, the distortion of the underlying geography in flow maps is often still done manually.

The presented flow map problem can be modeled as a graph drawing problem where the edges must avoid certain obstacles. A graph is a mathematical structure consisting of a set of objects, called vertices or nodes, and a set of relations between pairs of vertices, called edges. Typically, a graph is drawn in the plane using dots to represent vertices and curves connecting pairs of dots to represent edges. A graph is called *planar* if it can be drawn without edge crossings. Such a drawing, in which the edges only intersect at the endpoints, is called a *planar embedding* of the graph. In a weighted graph, each edge is assigned a weight to indicate some cost or load. The flow lines in a flow map can naturally be represented by a weighted graph with vertices at fixed positions on the map and edge weights indicating the amounts of flow being moved. Like flow maps, weighted graphs can be visualized by drawing each edge with a width, or thickness, proportional to its weight. The vertex sizes must be increased accordingly. When some of the thick edges (partially) overlap each other, the corresponding edge weights may not be accurately visualized, decreasing the readability of the graph (see Figure 1.3(a)). Therefore, ideally, the graph is drawn such that the edges only overlap at the endpoints (see Figure 1.3(b)).

We refer to the map features that we do not want to obscure as obstacles, and we assume that the input graph does not overlap any of them. If the edges describe specific routes between the vertices, it is important to preserve key features of their original routes when drawing the graph. In the presence of obstacles, a crucial property that we want to maintain is the way in which the edges wind around the obstacles. For instance, if an input edge passes underneath a certain obstacle, the corresponding (thick) edge in the graph drawing should also pass underneath that same obstacle. This principle is known as *path homotopy*. Informally, two paths with the same endpoints are homotopic if one can be 'continuously deformed' into the other without passing over any of the vertices or obstacles. A formal definition of path homotopy is given in Chapter 3.

We assume that the edge weights specify the thicknesses of the edges, and are interested in drawing a weighted graph with thick edges that avoids a set of obstacles, such that the thick edges only overlap at the endpoints and are homotopic to the input edges. Following the definition



(a) (b)

Figure 1.3: Two different embeddings of the same weighted graph, where the vertices have fixed positions. (a) A bad embedding in which edges cross and overlap each other. (b) A good embedding in which the edges only overlap at the endpoints.

of homotopy, we may attempt to continuously deform the input edges to obtain a valid solution. However, sometimes no feasible solution exists. In particular, if an edge is enclosed by two obstacles and is thicker than the space available between them, the edge will inevitably overlap at least one of the obstacles. Another consideration is that rerouting might significantly distort the edges and make the original routes unrecognizable. These issues can be addressed by allowing the obstacles to be distorted, leading to a trade-off between preserving original edge routes and preserving the original configuration of obstacles. We only focus on the problem of infeasible edge routing and aim to solve it by distorting the obstacles, while allowing any homotopic routing of the edges.

We restrict our study to a simplified version of the problem, where we only consider a set of point obstacles and aim to optimize their new positions. This removes the challenge of deforming polygonal obstacles, allowing us to focus solely on displacement as method for distortion. However, while this seems to greatly simplify the problem, the reduced version still proves difficult.

1.1 Formal problem definition

The input to our problem consists of a weighted graph $G = (V, E, \tau)$ and a set O of n_o disjoint point obstacles, where each obstacle $\omega \in O$ has a position $(x_\omega, y_\omega) \in \mathbb{R}^2$ in the plane. Additionally, we are given a planar embedding of G in which each vertex $v \in V$ is assigned a position $(x_v, y_v) \in \mathbb{R}^2$ and each edge is represented by a polygonal curve between the corresponding vertices disjoint from the obstacles. The goal is to find a new optimal position $(x'_\omega, y'_\omega) \in \mathbb{R}^2$ for each obstacle $\omega \in O$ and a *planar thick embedding* of G (formally defined in Chapter 3) such that (1) each vertex $v \in V$ remains in the same position (x_v, y_v), (2) each edge $e \in E$ has thickness $\tau(e)$, (3) the thick edges only overlap at the endpoints, and (4) the thick edges are homotopic to the input edges. In optimizing the new obstacle positions, the objective should be to minimize either the total or the maximum displacement over all obstacles in O. The total displacement is a good measure of distortion but often leads to a more complex optimization problem. On the other hand, minimization of the maximum obstacle displacement tends to distribute the distortion more evenly across the obstacles, preventing obstacles from being significantly displaced, but possibly resulting in unnecessary displacements. We refer to this problem as the *Minimum Displacement Graph Embedding* (MDGE) problem. See Figure 1.4 for an illustration of the MDGE problem.

The aim of this thesis is to investigate the MDGE problem from both a theoretical and practical perspective. First of all, we would like to study the computational complexity of the MDGE problem. This involves determining whether the problem can be solved in polynomial time or if it is NP-hard. To this end, we may establish complexity results for simplified variants of the problem and use those findings to infer the complexity of the original problem. In addition to a theoretical analysis, we focus on developing a practical solution to the MDGE problem. Our goal

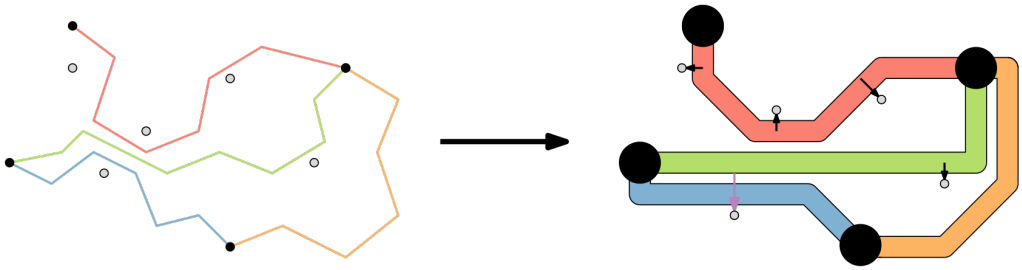


Figure 1.4: An illustration of the MDGE problem. In this example all edges have the same weight and the arrows indicate the displacements applied to the obstacles. If the goal is to minimize the total obstacle displacement, the solution has a cost equal to the combined length of the arrows. If the goal is to minimize the maximum obstacle displacement, the solution has a cost equal to the length of the purple arrow.

is to design an efficient algorithm that works well in practice. To demonstrate its feasibility, we aim to implement the proposed algorithm as a proof of concept. The objectives of our thesis are reflected by the following two research questions:

1. *What is the computational complexity of the MDGE problem?*
2. *How to develop a practical algorithm for the MDGE problem?*

1.2 Results and organization

We initially study a simplified version of the MDGE problem restricted to a single dimension, for which we provide an efficient algorithm running in linear time. In two dimensions, the MDGE problem is significantly more difficult. We prove that, if we allow the thick edges in the output to overlap, the MDGE problem is NP-hard, from which we conclude that the original MDGE problem is likely NP-hard as well. To solve the MDGE problem, we observe that we can model it as a constrained optimization problem, with constraints specifying the minimum required separation between each pair of obstacles. We show that by only enforcing constraints on the edges of a Delaunay triangulation on the obstacles and vertices, we obtain a 1.998-approximation of the minimum-separation constraints, while reducing the number of constraints from quadratic to linear. By imposing additional constraints to preserve the orthogonal order of the obstacles and vertices, we are able to efficiently compute the new obstacle positions using a linear program that approximates the optimal solution by a factor $\sqrt{2}$, under the assumption that the Delaunay triangulation remained the same after displacement. This linear program is combined with a method for computing thick homotopic edges to develop a practical algorithm for the MDGE problem.

While flow maps have been studied intensively, the MDGE problem, to the best of our knowledge, has not been subject to previous research. Therefore, in Chapter 2, we provide a broad overview of related work on the various components of the MDGE problem. Before presenting our findings, we introduce relevant concepts and definitions in Chapter 3. Chapter 4 is dedicated to the one-dimensional MDGE problem, after which we discuss NP-hardness in Chapter 5. We then present our algorithm for the MDGE problem in Chapter 6 and provide a brief experimental evaluation of our implementation in Chapter 7. Finally, we discuss our results in Chapter 8.

Chapter 2

Related work

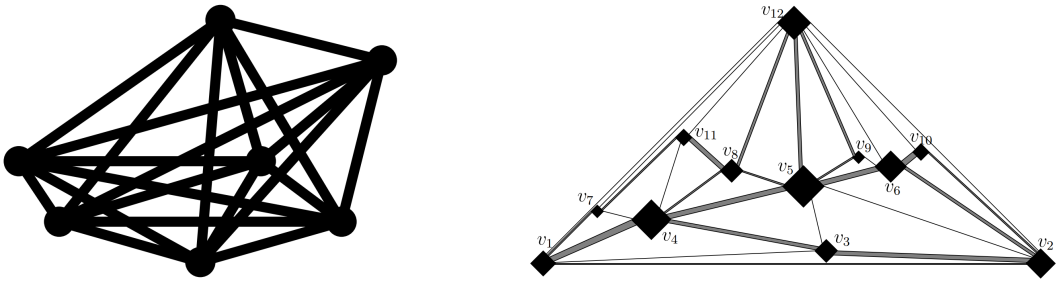
The MDGE problem consists of two main components: drawing graphs with thick edges that are homotopic to the input edges, and displacing the obstacles to admit a feasible solution. To the best of our knowledge, the combination of the two topics, the MDGE problem, has not been subject to previous research. In this chapter, we discuss related work on graph drawing and explore displacement problems encountered in the fields of motion planning and overlap removal that are similar to our problem of optimal obstacle displacement. To provide a broader context and better address real-world scenarios, we then extend our discussion of displacement problems to the more general problem of map deformation. Finally, we study the original motivation for the MDGE problem, namely flow maps.

2.1 Graph drawing

Graph drawing is a broad field of research in which many problems have been studied and many algorithms have been proposed [6, 35, 50, 83]. Typically, the goal is to produce clear and visually pleasing graph layouts that optimize certain quality measures. A common objective is to minimize the number of edge crossings in a graph drawing, as a high number of crossings decreases the understandability of the drawing [74]. The problem of determining whether a given graph can be drawn without edge crossings is a fundamental problem in graph drawing, known as the planarity testing problem. In 1974, Hopcroft and Tarjan [43] were the first to introduce a planarity testing algorithm that runs in linear time, which is asymptotically optimal. Closely related is the problem of finding a planar embedding of a graph, for which many methods exist [67].

Theoretical representations of graphs regard vertices as zero-dimensional points and edges as one-dimensional lines. In practice, however, graphs are drawn in the two-dimensional plane, which means that the vertices must have a certain size and the edges a certain thickness to ensure visibility. There are relatively few results on drawing graphs with thick edges. One of the challenges in drawing such graphs is to prevent ambiguity, that is, to keep vertices and edges visible and distinguishable. Van Kreveld [54] introduces several criteria to assess the quality of a bold graph drawing, which is a drawing in which vertices are drawn as black solid disks and edges as black thick line segments (see Figure 2.1(a)). Pach [71] argues that there exists an unambiguous bold drawing for every graph. When all the edges of a graph are drawn with the same thickness, they do not communicate any additional information besides connectivity. Barequet et al. [3] study the problem of finding a planar straight-line drawing of a weighted graph (see Figure 2.1(b)). The edges have thickness proportional to their weight and the vertices are drawn as solid diamonds with size proportional to the total weight of their incident edges.

The studies discussed so far allow an arbitrary placement of the vertices, while our problem requires the vertices to have fixed positions. A problem that does enforce this requirement, and that is closest to the MDGE problem in existing literature, is the General Fat Edge Drawing (GFED) problem introduced by Duncan et al. [23]. In contrast to the MDGE problem, the GFED problem



(a) A bold graph drawing. Adapted from [54].

(b) A planar straight-line weighted graph drawing. Adapted from [3].

Figure 2.1: Graph drawings with large vertices and thick edges.

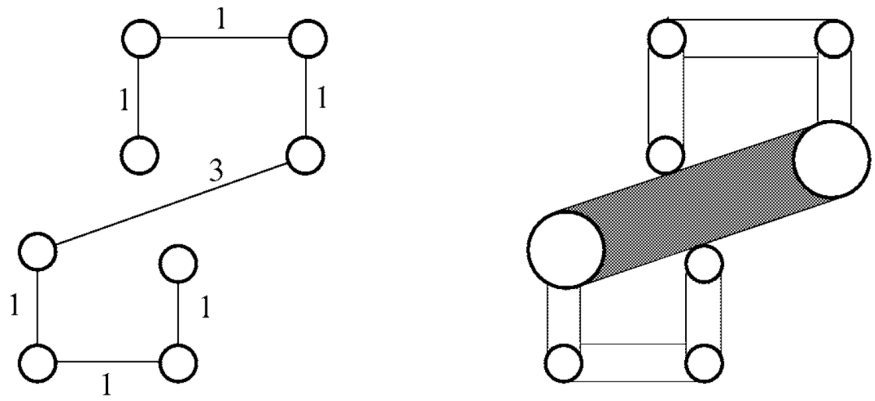


Figure 2.2: An illustration of the GFED problem, with input on the left and output on the right. The numbers next to the edges indicate their weight. The growing process terminated because the edges could not be grown any further while remaining interior disjoint. Adapted from [23].

considers the point obstacles to be fixed. The goal is to construct a homotopically equivalent planar drawing of a weighted graph in which the edges have maximum thickness proportional to their weight (see Figure 2.2). The paper shows how to solve a simplified version of the GFED problem in which the input graph has a maximum degree of 1, called the Fat Edge Drawing (FED) problem. Given a set of n shortest paths, the algorithm grows the paths simultaneously in thickness over time, at a speed proportional to the individual weight of each path. Throughout this growth process, the sum of the lengths of all paths remains as small as possible, and the homotopy among paths is preserved. The process terminates when it becomes impossible to increase the thickness of the paths while keeping them interior disjoint. Their algorithm runs in $\mathcal{O}(n^3 + k)$ time and uses $\mathcal{O}(n + k)$ space, where k is the maximum of the input and output complexities of all paths. The authors of the paper show how the algorithm can be extended to solve the GFED problem without increasing its time and space complexity.

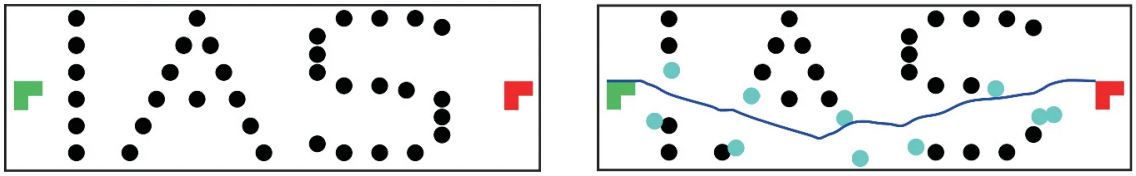
The main aspect in which our problem differs from the GFED problem is that we allow the obstacles to be displaced. Therefore, we may solve the MDGE problem by combining the growing algorithm by Duncan et al. with a method for displacing the obstacles. Instead of maximizing the thickness of the edges, we can simply terminate the growing process if the edges have reached their required thickness. Speckmann and Verbeek [78] also use this strategy and further adapt the algorithm to solve a modified version of the FED problem in which the line segments of the obstacles use a restricted set $\mathcal{C}$ of orientations. They study the NP-hard optimization problem of computing minimum-link $\mathcal{C}$-oriented thick paths that are homotopic to the input paths and provide an efficient $\mathcal{O}(|\mathcal{C}|)$-approximation algorithm for it.

2.2 Motion planning

The classical motion planning problem asks to find a path from a given start position to a given destination while avoiding a set of fixed obstacles. However, if no such path exists, it is necessary to move some of the obstacles in order to yield a feasible path. The Navigation Among Movable Obstacles (NAMO) problem solves this issue by allowing the moving entity to move obstacles along its way to clear the path. The goal is to minimize the work done by the robot, which is typically defined in terms of number of obstacles moved, total obstacle displacement or traversed path length. NAMO is a well-studied problem that is proven to be NP-hard [96] and for which many motion planners have been proposed [7, 13, 66, 81].

The robot may not always be capable of moving obstacles, in which case obstacle reconfiguration and path construction must be performed separately. Hauser [40] introduces the Minimum Constraint Displacement (MCD) motion planning problem, where the goal is to minimize the displacement of a set of obstacles, also called constraints, to allow for the construction of a feasible path. The problem is proven to be NP-hard, even in the discrete version where the configuration space is restricted to a graph. The MCD problem allows the displaced obstacles to overlap, which is often not desired. A more practical motion planner was presented by Thomas and Mastrogiovanni [84], which ensures that the obstacles are disjoint after displacement. Their algorithm first computes a feasible path for the robot that minimizes obstacle displacement during the planning phase (see Figure 2.3), after which the refinement phase removes new overlaps among the displaced obstacles using an iterative displacement technique.

Although the MDGE problem does not directly involve motion planning tasks, it shares many similarities with motion planning problems. In fact, we can naturally model the MDGE problem as a multi-robot minimum-displacement motion planning problem. To see this, consider a planar embedding of a graph $G = (V, E, w)$ and a set of point obstacles O given as input to the MDGE problem. We assign to each edge $e \in E$ a disk with diameter $\tau(e)$, representing a robot. The goal is then to find new positions for the obstacles in O with minimal displacement and a set



(a) An L-shaped robot with the task of moving from the green position to the red position while avoiding a set of circular obstacles.

(b) The output of the planning phase of the motion planner by Thomas and Mastrogiovanni [84], where the displaced obstacles are shown in cyan.

Figure 2.3: Minimum-displacement motion planning. Adapted from [84].

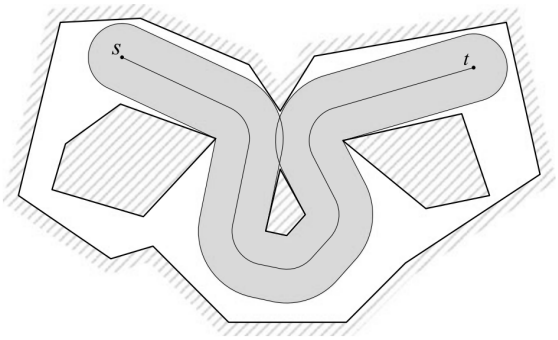


Figure 2.4: A motion planning instance for which only a self-overlapping thick path from s to t exists. Adapted from [52].

of collision-free paths for the robots, such that each robot corresponding to the edge $(u, v) \in E$ moves from u to v, and the paths are homotopic to the input paths. Note that the paths must be collision-free but may have common start and end positions.

The presented motion planning variant of the MDGE problem may be seen as a generalization of the motion planning problem studied by Chew [15], which asks for the shortest path for a disk-shaped robot avoiding a set of polygonal obstacles. Chew provides a polynomial-time algorithm that 'inflates' the obstacles according to the size of the robot and then simply computes the shortest path using Dijkstra's algorithm. However, the constructed thick path may self-overlap [44, 52] (see Figure 2.4), which is typically not an issue for a moving robot, but is not allowed in a solution to the MDGE problem. Kostitsyna and Polishchuk [52] show that it is NP-hard even to *decide* whether there exists a feasible thick path without self-overlap.

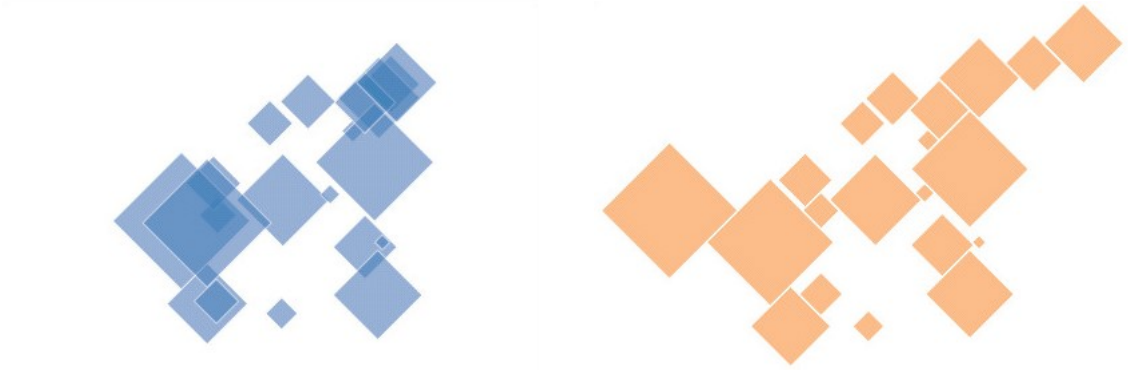
Motion planning typically focuses on the feasibility of paths without imposing constraints on their homotopy. As a result, to the best of our knowledge, the described motion planning equivalent of the MDGE problem has not been studied before.

2.3 Overlap removal

Given a set of overlapping objects in the plane, a fundamental geometric problem is to construct a new layout for the objects without overlap. The overlap removal problem has a wide range of applications, such as in proportional symbol maps or node overlap removal in graph drawing, but unfortunately most variants are NP-hard. In fact, for a set of rectangles, both the problems of minimum-area layout [41] and minimum-displacement overlap removal [32] have been shown to be NP-hard. If no constraints are enforced on the orthogonal order, minimizing the maximum displacement is NP-hard for circles, disks and diamonds [27].

Many methods have been proposed for the overlap removal problem [12]. Due to its complexity, various heuristics have been introduced [32, 41, 46, 65]. One of the most prominent algorithms is PRISM [30], which removes overlap using a proximity graph of the input but cannot guarantee that the orthogonal order of the objects is preserved. Another common strategy involves constrained optimization [24, 42, 63].

Meulemans [64] argues that changing the symbol size from a square to a diamond significantly reduces the complexity of the minimum-displacement overlap removal problem. They propose a linear program with orthogonal order constraints to efficiently solve the problem (see Figure 2.5). The constraints on the orthogonal order ensure that the x- and y-orders on the centers of the diamonds are maintained. The constraint that two diamonds must be disjoint can be specified



(a) A set of overlapping diamonds.

(b) The output of the linear program by Meulemans [64], removing overlap using minimum displacement while maintaining the orthogonal order.

Figure 2.5: Minimum-displacement overlap removal of a set of diamonds. Adapted from [64].

using the Manhattan distance, as its unit circle is exactly a diamond. The objective function of the linear program minimizes the total displacement of the diamonds. While the Euclidean distance is the most natural distance metric, it is non-linear and thus cannot directly be used in the linear program. Instead, Meulemans measures the displacement of the diamonds in the objective function using a convex polyhedral distance function, whose unit 'circle' forms a convex polygon centered around the origin. It can be used to arbitrarily well approximate the unit circle of the Euclidean distance. Specifically, a distance function whose unit circle consists of a k-gon with $k = \mathcal{O}(\epsilon^{-1/2})$ yields a $(1+\epsilon)$-approximation of the problem under the Euclidean distance. The reason for this problem being tractable in contrast to its NP-hard counterparts involving other shapes, is that the orthogonality constraints induce a convex feasible space for placing a diamond with respect to another, which in general does not hold for other shapes.

As will be shown in Chapter 6, the MDGE problem can be modeled as a constrained optimization problem, where the constraints specify the minimum required separation between pairs of obstacles. Since these constraints closely resemble the disjointness constraints used for overlap removal, we may adapt constrained optimization techniques for minimum-displacement overlap removal, such as the linear program by Meulemans, to compute the new obstacle positions for the MDGE problem.

2.4 Map deformation

While the MDGE problem only considers a set of point obstacles, practical applications, such as flow maps, often involve polygonal obstacles. The space that is available for visualizing thematic information then not only depends on the positioning of the obstacles, but also on their shapes. As a result, displacement alone may not always be sufficient as sole method for distortion, requiring more advanced deformation techniques. We provide a broad overview of map deformation methods, which are not directly applicable to the MDGE problem but may become relevant when extending the problem towards actual flow map construction. The underlying geography of a map may be distorted for different purposes. We distinguish between applications of map deformation that focus on emphasizing information, and those aimed at reducing cartographic detail.

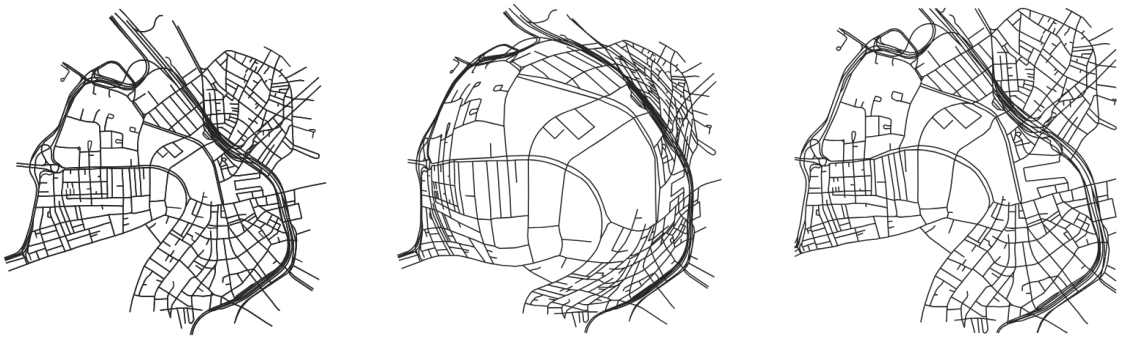
2.4.1 Emphasizing information

Focus maps

In a focus map, also called variable-scale map, one or more geographic regions are enlarged to help users focus on important or interesting map features. The enlargement of the focus area(s) leaves less space for displaying the surrounding context, which must therefore be distorted. Focus maps are particularly useful for personal navigation by providing detailed cartographic data close to the user's position, while still showing the environment sufficiently to help the user orient himself.

Furnas [29] first introduced the concept of the fisheye view, which is a focus map in which the scale simply decreases linearly from a given scale at one point, the focus point, towards the boundaries of the map (see Figure 2.6(b)). This creates a magnifying glass effect where the map becomes increasingly distorted further away from the focus point. Fairbairn and Taylor [26] generalize this approach to construct a fisheye view with scale changing in a linear fashion from a given scale at the focus point to a given scale at one or more specified points.

The main drawback of a fisheye view is that the distortions are distributed across the entire map. Additionally, due to the decreasing scale, there is a high density of geographic details near the borders of the map, making these parts of the map difficult to read. These issues are mitigated by focus+glue+context maps [14, 37, 101, 102], which have a designated glue area around the focus area to absorb all distortion (see Figure 2.7(b)). In the glue area, the scale is linearly decreased from the scale of the focus area to the scale of the context area. The global angles from the focus point are preserved, which means that straight lines originating from the focus point remain straight. This is a crucial property in personal navigation, as it ensures that a user located at the focus point observes the correct angles from his position.

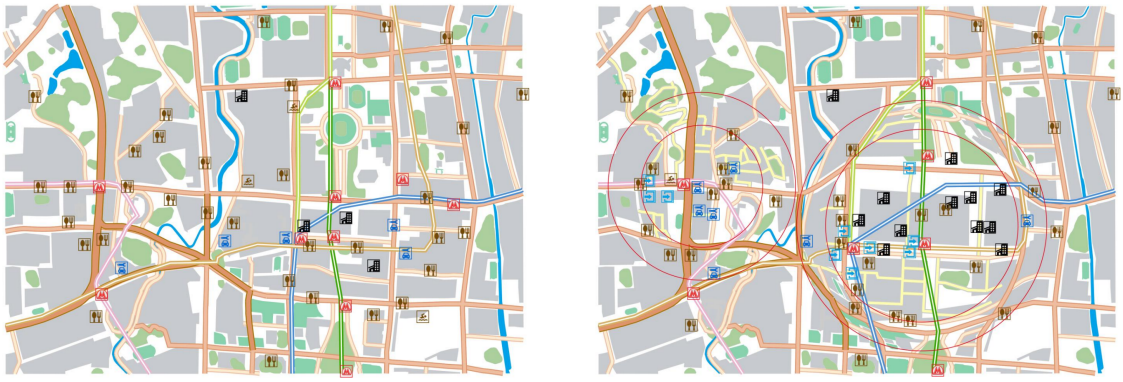


(a) The original map.

(b) A fisheye view, computed using the method by Yamamoto et al. [101].

(c) An optimization-based focus map, computed using the method by Haunert and Sering [39].

Figure 2.6: A fisheye view and an optimization-based focus map of Boston with focus on the center of the map. Adapted from [21].



(a) The original map.

(b) A focus+glue+context map with focus and glue areas indicated in red, computed using the method of Zhao et al. [102].

Figure 2.7: A focus+glue+context map of Shenzhen, China. Adapted from [102].

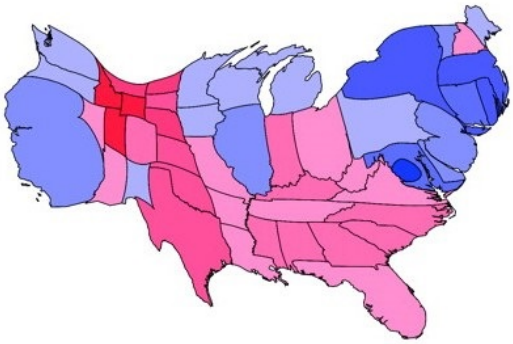
Fisheye views and focus+glue+context maps can be easily generated using a mapping function that simply scales the map in a predefined manner. As a result, the distortions may be applied to dense areas and be highly concentrated in certain regions, such as near the boundaries of fisheye views and in the glue area of focus+glue+context maps. This may severely deteriorate the readability and recognizability of the focus map, which can be mitigated by optimizing the distribution of distortion, particularly by moving the distortion towards sparse areas [58, 87]. In addition to these optimization techniques for redistributing the distortion in existing focus maps, various methods have been proposed for the construction of focus maps with minimum distortion [21, 39] (see Figure 2.6(c)).

The reason for enlarging certain regions in focus maps is to create more space for displaying the corresponding map features. However, by instead enlarging the area in between different map features, more 'empty' space can be created between them, accommodating the visualization of thematic information. Thus, by employing focus map techniques in the construction of flow maps, more space can be generated for visualizing the flow lines. However, the methods generally do not provide control over the distortion of individual map features and may move the distortion towards unwanted areas.

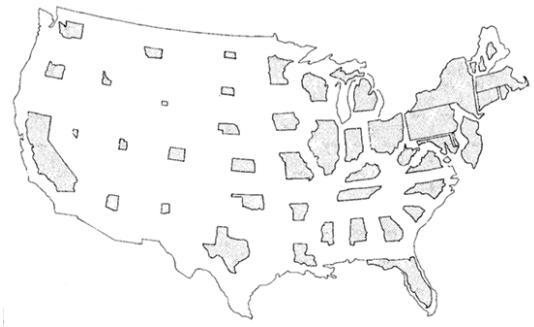
Cartograms

A cartogram is a thematic map in which geographic regions, such as countries or provinces, are distorted such that their area is proportional to the value of a certain variable, such as population or average income, in the corresponding region. While the modified areas should accurately communicate the statistical information (statistical accuracy), the regions should also sufficiently maintain their original geographic shapes and relative positions (geographical accuracy), as well as their original adjacencies (topological accuracy). The quality of a cartogram is typically evaluated according to these three performance measures, and trade-offs must be made between them in the construction of cartograms. Many different types of cartograms have been proposed in literature [69, 88], optimizing different performance measures [2, 68]. They can be categorized into four types: contiguous, non-contiguous, Dorling and rectangular (see Figure 2.8). Some cartograms achieve perfect statistical accuracy at the expense of distorting the original shapes of regions (Dorling) or the topology of the map (Dorling and non-contiguous). Others allow for some statistical errors to better preserve geographical (contiguous) or topological accuracy (contiguous and rectangular).

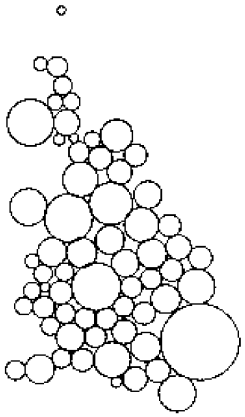
Alam et al. [2] and Nusrat et al. [68] discuss various methods for measuring the statistical, geographical and topological accuracy of cartograms. Statistical accuracy is usually measured by the average or maximum cartographic error over all regions, which is the deviation of the area of a modified region from the area required by the corresponding statistic. The shape preservation



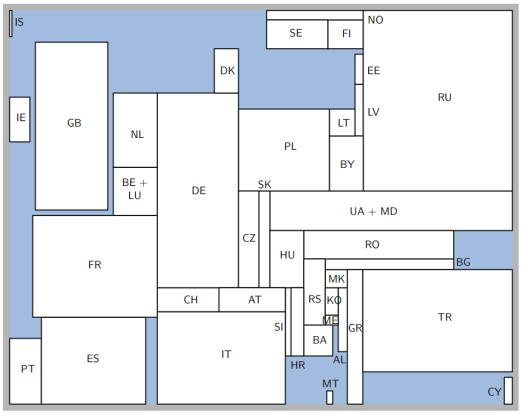
(a) A diffusion-based contiguous cartogram showing the number of electors of the United States in the presidential election results of 2000 (shades of red and blue indicate Republican and Democrat state majorities, respectively). Adapted from [34].



(b) A non-contiguous cartogram showing the population of the United States with 65 years of age and over in 1970. Adapted from [70].



(c) A Dorling cartogram showing the population of Great Britain in 1996. Adapted from [22].



(d) A rectangular cartogram showing the population of Europe in 2011. Adapted from [11].

Figure 2.8: The four different types of cartograms.

of a region can be measured by the Hamming distance, also known as the symmetric difference, between the original and modified region, which is the fraction of the area in exactly one of the regions when they are normalized and placed on top of each other. Another shape similarity measure is the difference in aspect ratios of the axis-aligned bounding boxes. Alam et al. [2] conclude from their experiments that the Hamming distance generally best captures the similarity between shapes of regions. The preservation of the relative positions of the regions can be assessed using the angular orientation error, which is the average change in the slope of the line through the centroids of pairs of regions. Finally, the topological accuracy of a cartogram is typically measured by the fraction of the regional adjacencies that the cartogram manages to preserve.

Our reason for applying distortion in flow maps is to avoid cartographic conflicts, rather than to communicate statistical information. Therefore, we do not have a notion of statistical accuracy in flow maps. The geographical accuracy of the map features must however be preserved, thus the corresponding measures used in cartogram design may be used to assess the quality of map deformation. If it is also required to maintain the adjacencies between the obstacles, the topological accuracy measures may prove useful.

2.4.2 Reducing cartographic detail

Map schematization

Schematization is a well-studied cartographic technique for simplifying and abstracting geographical and spatial information to improve the readability of a map. A common application of schematization is in transit maps, which provide a simplified view of public transportation systems such as train or metro networks (see Figure 2.9). To ease navigation through the network, the routes between stations are schematized and the stations are evenly distributed across the map. Many methods have been proposed for the construction of transit maps [98, 99]. To further abstract from non-essential details, the underlying geography of schematic maps is typically also schematized [4, 17, 20].

Van Dijk et al. [19] present a method for the partial schematization of focus maps, which can be applied in various ways to obtain maps for different purposes. For instance, by uniformly schematizing the context in a focus map, the focus area is emphasized, making it easier for the user to locate the focus area on the map. Alternatively, by gradually increasing the schematization further away from the focus area, the schematized focus map again actively draws the attention of the user to the focus area but at the same time maintains the relationships of the focus area with its surroundings. Conversely, techniques used in the construction of focus maps may be used to enhance schematic maps. This involves tasks such as enlarging congested areas [85], redistributing line density [86], or highlighting certain routes in metro maps [92].

Figure 2.9: An octilinear transit map of the London Underground. Adapted from [75].

Map generalization

When reducing a larger-scale map to a smaller-scale map, less space remains available for visualizing detailed cartographic information. This may cause map features to become too close to each other or even overlap, referred to as spatial conflicts. Therefore, the level of cartographic detail needs to be adjusted to match the presented scale of the map. This process, known as map generalization, finds its main applications in maps involving varying scales, such as focus maps or zoomable maps used in navigation systems. Map generalization can be established through the application of various types of operators, including selection, simplification, aggregation and displacement of map features, for which many algorithms have been proposed [57, 79, 95]. Since addressing all generalization operators would be beyond the scope of this study, we solely focus on displacement as it is the only operator directly relevant to the MDGE problem.

Displacement is a common generalization technique for resolving spatial conflicts (see Figure 2.10), with a distinction between sequential and global displacement methods. Sequential displacement algorithms move map features one by one and in a particular order. Ruas [76] starts with the object causing the most significant conflict, moves it to an appropriate position, and reactively solves newly created conflicts. Ai et al. [1] first partition the buildings on a map using a Delaunay triangulation and then compute a vector field, indicating the direction and magnitude of the displacement force exerted on the buildings. An alternative strategy is to use a Voronoi diagram to group buildings into zones and iteratively displace them [5, 77].

In sequential methods, the displacement of one object may introduce new conflicts between other objects. Global displacement algorithms address this issue by considering all features on the map, and thus all spatial conflicts, simultaneously. Mackaness and Purves [62] simply fit a circle around each object, consider a set of candidate locations for it and move the objects such that overlap between the circles is minimized. More advanced combinatorial optimization techniques used for displacement include steepest gradient descent [61, 93], simulated annealing [93], tabu search [94], partical swarm optimization [45] and genetic algorithms [56, 60, 73, 82, 97]. Another global displacement strategy involves specifying constraints on a solution to the problem and then employing optimization to obtain the necessary movement for each object [38, 47].

It may not always be feasible to resolve all spatial conflicts solely through the displacement of map features, especially in dense areas. Therefore, addressing the remaining conflicts requires the deployment of additional operators in the generalization process. Map generalization aims to reduce overlap among map features, which is also the reason for distortion in flow maps. Therefore, the corresponding displacement techniques seem useful for distorting the underlying geography of

(a) The original map.

(b) A generalized map where spatial conflicts are resolved through displacement, computed using the immune genetic algorithm by Sun et al. [82].

Figure 2.10: Generalization of the Jiang'an District in Wuhan, China. Adapted from [82].

flow maps. However, most other generalization operators, such as selection and aggregation, are not applicable to the problem we are studying, as we do not consider those types of deformation.

2.5 Flow maps

Flow maps form the main application of the MDGE problem. They have been studied intensively and many variants exist. Early automated methods for constructing flow maps [28, 89, 90] simply depict flows using straight lines, leading to visual clutter (see Figure 1.1). These maps, known as origin-destination flow maps, focus on showing the existence of a connection rather than the route of the flow. More recent approaches [36, 48, 53, 80] are able to reduce visual clutter by adhering to certain design principles [49], such as the use of curved flow lines and the minimization of overlap between them (see Figure 2.11).

Although the compliance with design principles improves the readability of the map, origin-destination flow maps often still suffer from visual clutter due to the typically large number of crossings. This issue is mitigated by distributive flow maps, which bundle flow lines with the same origin into a single line with thickness proportional to the combined flow (see Figure 1.2). Flows are distributed across the map by gradually branching the flow line into smaller flow lines that are then directed towards their respective destination. Phan et al. [72] employ hierarchical clustering to construct single-source distributive flow maps. Their method minimizes the number of crossings, but does not guarantee a crossing-free map and does not prevent the obstruction of map features. Another approach by Buchin et al. [10] uses spiral trees to generate high-quality distributive flow maps (see Figure 2.12). Their algorithm ensures that the flow lines are crossing-free and avoid specified obstacles such as landmasses. Distributive techniques are particularly useful for constructing single-source flow maps, as in flow maps involving multiple sources crossings are typically unavoidable.

Automated methods for flow map construction typically do not consider map deformation, and in flow maps that do contain a distorted representation of the underlying geography, the distortion is often performed manually. Van Dijk et al. [19] propose the use of focus map techniques to make space for thematic information (see Figure 2.13). In particular, they first add bottleneck edges to the map to obtain a connected subdivision and control its rigidity, and then use the focus map method by Haunert and Sering [39] to enlarge the endpoints of bottlenecks between objects to be separated. Their approach successfully distorts the map to allow clear flow communication while maintaining geographical accuracy, but does not provide any guarantee on the quality of the solution. Overall, we can conclude that the study of map deformation for flow map construction is a relatively unexplored area in existing algorithmic literature, highlighting the relevance of the MDGE problem.

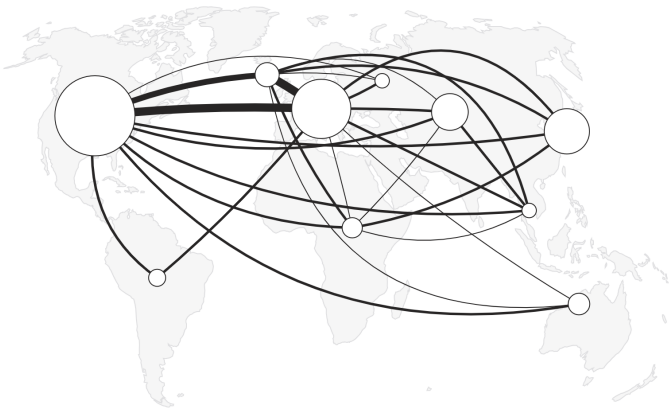


Figure 2.11: A flow map showing international investments. Adapted from [48].

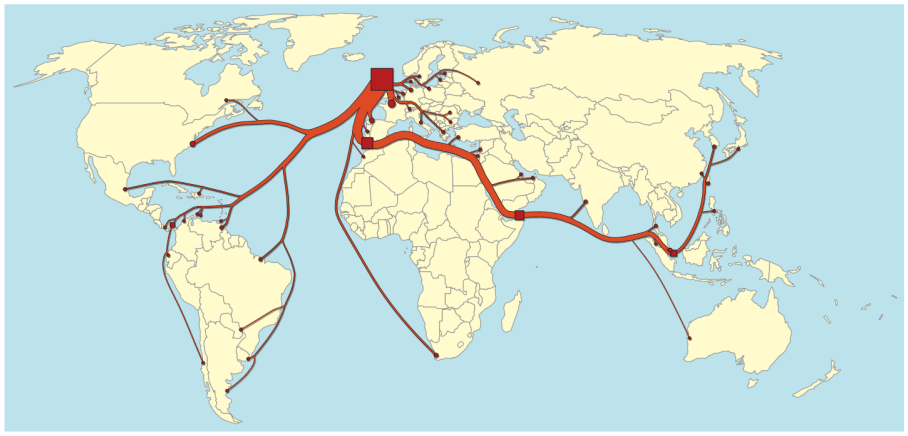


Figure 2.12: A distributive flow map showing the top 50 whisky exports by volume from Scotland in 2009. Adapted from [10].

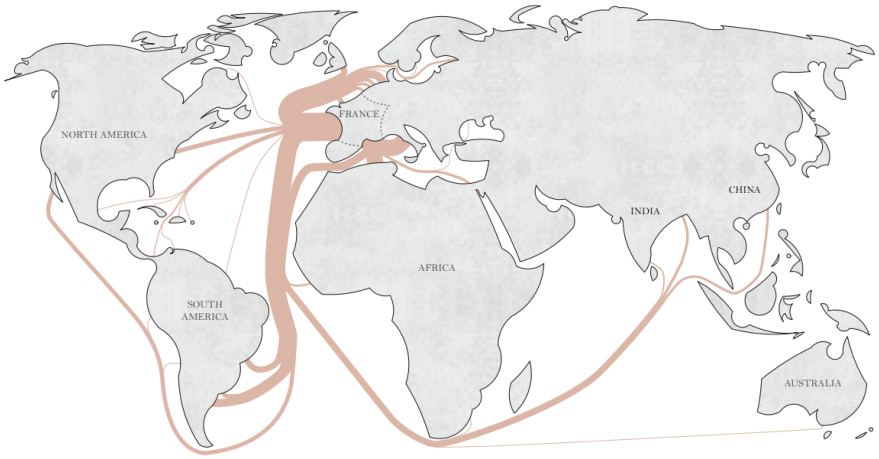


Figure 2.13: A distorted schematic version of Minard's flow map. Focus map techniques are used to make space for the thematic information, while schematization removes unnecessary detail. Adapted from [19].

Chapter 3

Preliminaries

Homotopy

We first provide a formal definition of path homotopy introduced by Gao et al. [31]. Define a closed set $\mathcal{B}$ consisting of all vertices in V and all obstacles in O. Let $\pi, \pi' : [0,1] \to \mathbb{R}^2$ be two continuous paths with the same endpoints that do not contain an element of $\mathcal{B}$ except for their endpoints. Then π and π' are *homotopic* with respect to $\mathcal{B}$ if there exists a continuous function $H : [0,1] \times [0,1] \to \mathbb{R}^2$ with the following properties:

1. $H(0,\mu) = \pi(\mu)$ and $H(1,\mu) = \pi'(\mu)$ for all $0 \leq \mu \leq 1$.
2. $H(\lambda,0) = \pi(0) = \pi'(0)$ and $H(\lambda,1) = \pi(1) = \pi'(1)$ for all $0 \leq \lambda \leq 1$.
3. $H(\lambda,\mu) \notin \mathcal{B}$ for all $0 \leq \lambda \leq 1$ and $0 < \mu < 1$.

The first two properties formalize the continuous deformation from π to π', while the third property ensures that no vertices or obstacles are passed during the deformation.

The given definition of homotopy assumes that the obstacles in O are fixed, while the MDGE problem allows them to be displaced. Therefore, we need to adapt the definition to capture the concept of homotopy under displaced obstacles. Our approach is to model the displacement of each obstacle as a function of time, and alter the definition of H such that it does not pass over any of the vertices or obstacles at any point in time. The chosen trajectories along which the obstacles move highly affect the final solution. As can be seen in Figure 3.1, having insufficient restrictions on the movement of the obstacles may lead to loss of the original homotopy. In this example, the obstacles are moved along a curve to avoid the edge and eventually return to their original position. Since the obstacles have retained their original position, we would expect that the homotopy remained the same. However, this is clearly not the case, as the orientations of the obstacles with respect to the edge have changed significantly. This causes the edge to wind around the obstacles in a completely different way than it did initially, thereby changing the homotopy.

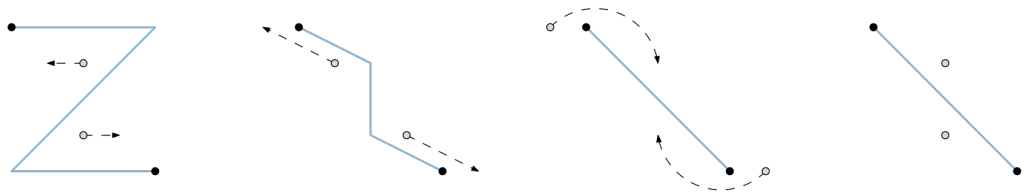


Figure 3.1: An example for which obstacle displacement with insufficient restrictions on the trajectories leads to loss of homotopy. From left to right, the obstacles are moved along a path that eventually returns to their original position, and the edge is deformed but remains homotopically equivalent. The edge and obstacles remain disjoint. While the obstacles retained their original positions, their orientations with respect to the edge have changed significantly.

We propose to represent the displacement of each obstacle $\omega \in O$ by the linear interpolation $f_\omega(t) = ((1-t)x_\omega + tx'_\omega, (1-t)y_\omega + ty'_\omega)$ for $0 \leq t \leq 1$. By restricting the obstacles to move along a straight line, we prevent them from moving 'around' vertices, thereby better preserving their original orientations with respect to the edges. Let $\mathcal{B}(t)$ denote the closed set consisting of all vertices in V and all obstacles in O at time t, that is, $\mathcal{B}(t) = V \cup \{f_\omega(t) \mid \omega \in O\}$ for all $0 \leq t \leq 1$. To extend the definition of homotopy towards the setting with displaced obstacles, we simply replace the third condition by the following.

3. $H(\lambda, \mu) \notin \mathcal{B}(\lambda)$ for all $0 \leq \lambda \leq 1$ and $0 < \mu < 1$.

While the original condition only considered the initial configuration of obstacles, the new condition specifies that H does not intersect any of the displaced obstacles during the deformation at any point in time. We now say that π and π' are *homotopic under displacement*, or simply homotopic for convenience, and use this version of homotopy for the MDGE problem. Path homotopy describes an equivalence relation, which means that every path is part of an equivalence class, called *homotopy class*. The homotopic relation requires the paths to be disjoint from the vertices and obstacles, but we allow obstacles to lie on the boundary of a path. To this end, we extend every homotopy class to its closure.

Planar thick embedding

Let $\pi : [0,1] \to \mathbb{R}^2$ be a continuous path. The *thick path* with *spine* π and thickness $\Delta > 0$ is defined as the Minkowski sum $\mathcal{S}_\pi^\Delta = \pi \oplus D_{\Delta/2} = \{p + q \mid p \in \pi, q \in D_{\Delta/2}\}$, where D_r denotes a disk of radius $r > 0$ centered at the origin. Note that two thick paths $\mathcal{S}_\pi^\Delta$ and $\mathcal{S}_{\pi'}^{\Delta'}$ are *interior disjoint* if the distance $d(p,q)$ between any two points on the spines $p \in \pi$ and $q \in \pi'$ is at least $\frac{\Delta+\Delta'}{2}$. Otherwise, the thick paths are said to *overlap*. For notational convenience, we also refer to a thick path $\mathcal{S}_\pi^\Delta$ by π.

Let $E = \{e_1, \ldots, e_m\}$ be the set of edges given as input to the MDGE problem. Each edge $e_i = \{u, v\} \in E$ with $1 \leq i \leq m$ is embedded in the plane as a polygonal curve, which can be represented by a continuous path $\pi_i : [0,1] \to \mathbb{R}^2$ with $\pi_i(0) = u$ and $\pi_i(1) = v$. We define a *planar thick homotopic embedding under displacement* of graph $G = (V, E, \tau)$, or simply *planar thick embedding* for convenience, as an embedding of G with the following properties:

- Each vertex $v \in V$ is represented by a simply connected closed region P_v.
- Each edge $e_i \in E$ is represented by a thick path $\mathcal{S}_{\pi'_i}^{\tau(e_i)}$ with spine $\pi'_i : [0,1] \to \mathbb{R}^2$, where π_i and π'_i are homotopic with respect to $\mathcal{B}$ under displacement.
- For each pair of distinct vertices $u, v \in V$, the two corresponding regions are interior disjoint, that is, $\text{int}(P_u) \cap \text{int}(P_v) = \emptyset$.
- For each pair of distinct non-incident edges $e_i, e_j \in E$, the corresponding thick paths are interior disjoint, that is, $\text{int}(\mathcal{S}_{\pi'_i}^{\tau(e_i)}) \cap \text{int}(\mathcal{S}_{\pi'_j}^{\tau(e_j)}) = \emptyset$.
- For each pair of distinct edges $e_i, e_j \in E$ incident on the same vertex $v \in V$, the corresponding thick paths may only overlap at the endpoint v, that is, $\text{int}(\mathcal{S}_{\pi'_i}^{\tau(e_i)}) \cap \text{int}(\mathcal{S}_{\pi'_j}^{\tau(e_j)}) \subseteq P_v$.
- For each vertex $v \in V$ and edge $e_i \in E$, the corresponding vertex region and thick path may only overlap, that is, $\text{int}(P_v) \cap \text{int}(\mathcal{S}_{\pi'_i}^{\tau(e_i)}) \neq \emptyset$, if and only if e_i is incident on v.
- For each vertex $v \in V$, the corresponding region does not contain any of the displaced obstacles, that is, $(x'_\omega, y'_\omega) \notin \text{int}(P_v)$ for all $\omega \in O$.
- For each edge $e_i \in E$, the corresponding thick path does not contain any of the displaced obstacles, that is, $(x'_\omega, y'_\omega) \notin \text{int}(\mathcal{S}_{\pi'_i}^{\tau(e_i)})$ for all $\omega \in O$.

Chapter 4

The one-dimensional problem

In this chapter, we study the one-dimensional MDGE problem, which is a simplified version of the MDGE problem restricted to a single dimension. To accommodate the representation of edge thickness in one-dimensional space, we first adapt the original problem definition. Subsequently, we present an algorithm to optimally solve the one-dimensional MDGE problem in linear time.

4.1 Problem definition

The input to the one-dimensional MDGE problem consists of a set of weighted edges E and a set O of n_o disjoint point obstacles, where each obstacle $\omega \in O$ has a position $x_\omega \in \mathbb{R}$. Each edge $e \in E$ is represented as a single point and assigned a position $x_e \in \mathbb{R}$ disjoint from the obstacles. Note that we no longer consider a graph as part of the input, but each edge can still be interpreted as connecting two implicit vertices, thereby representing a graph structure. In one-dimensional space, a thick edge $e \in E$ is represented as a line segment of length $\tau(e)$. The goal is to find a new optimal position $x'_\omega \in \mathbb{R}$ for each obstacle $\omega \in O$ and a line segment l_e for each edge $e \in E$ such that (1) each line segment l_e has length $\tau(e)$, (2) the line segments only intersect at their endpoints, and (3) the thick edges are homotopic to the input edges. Here, homotopy is naturally defined as preserving the x-order of the obstacles and edges. See Figure 4.1 for an illustration of the one-dimensional MDGE problem.

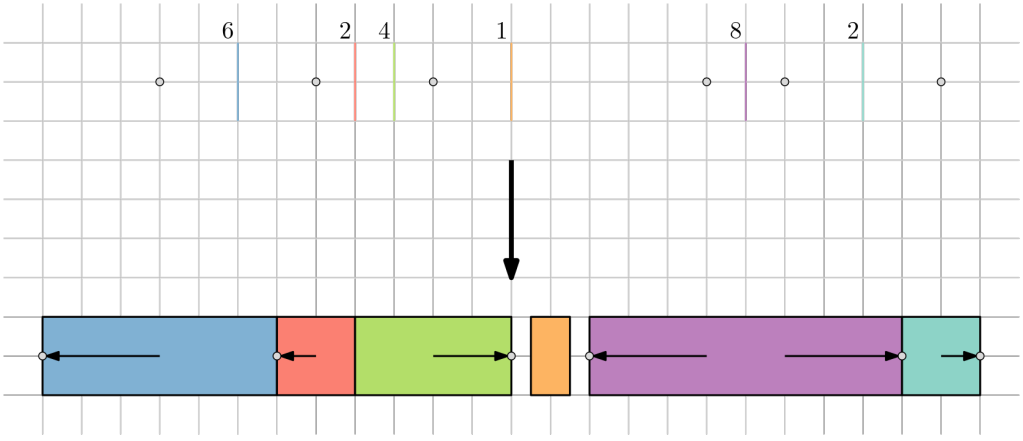


Figure 4.1: An illustration of the one-dimensional MDGE problem, where the goal is to minimize the maximum obstacle displacement. Note that the edges are visualized as vertical line segments but are actually points. The numbers above the edges indicate their thickness and the arrows the displacements applied to the obstacles. The solution has a cost of 3, which is optimal.

Let $O = \{\omega_1, \ldots, \omega_{n_o}\}$ be the set of obstacles, sorted by x-coordinate. For ease of notation, we use x_i instead of x_{ω_i} to denote the x-coordinate of obstacle ω_i for all $1 \leq i \leq n_o$. Furthermore, let $E_{i,j} = \{e \in E \mid x_i < x_e < x_j\}$ be the set of edges located in between obstacles ω_i and ω_j for $1 \leq i \leq j \leq n_o$, and let their combined edge thickness be denoted by $\tau(E_{i,j}) = \sum_{e \in E_{i,j}} \tau(e)$.

4.2 The algorithm

We provide an efficient algorithm to solve the one-dimensional MDGE problem, where the objective is to minimize the maximum obstacle displacement. To achieve this, we first determine the smallest maximum displacement that allows for a valid solution. For all $1 \leq i \leq n_o$, let $D[i]$ be defined as

$$D[i] = \begin{cases} 0 & \text{if } i = 1 \\ \max(0, D[i-1] + \tau(E_{i-1,i}) - (x_i - x_{i-1})) & \text{otherwise.} \end{cases} \tag{4.1}$$

For a pair of obstacles ω_{i-1}, ω_i with $1 < i \leq n_o$, we know that $\tau(E_{i-1,i})$ is the amount of space needed to draw their separating thick edges, while $x_i - x_{i-1}$ is the distance between them. Hence, the difference $\tau(E_{i-1,i}) - (x_i - x_{i-1})$ equals the space deficit between the two obstacles. By adding $D[i-1]$, we also account for the space deficit between ω_{i-1} and any of the obstacles to the left of it. In other words, $D[i]$ denotes the maximum shortage of space between obstacle ω_i and any of its preceding obstacles ω_j with $1 \leq j < i$. Now let δ_{min} be defined as

$$\delta_{min} = \max_{1 \leq i \leq n_o} \frac{D[i]}{2}. \tag{4.2}$$

We claim that δ_{min} is the smallest maximum displacement of any valid solution to the problem. Given some displacement $\delta \geq 0$, our displacement strategy works as follows. Start with obstacle ω_1 and displace it by δ to the left. Now consider obstacle ω_2 and try to displace it by δ to the left as well. However, if this does not leave enough space for the thick edges in between ω_1 and ω_2, place ω_2 at an offset of $\tau(E_{1,2})$ to the right of ω_1's new position. Continue this process from left to right until all obstacles are displaced. The idea is that all obstacles are placed to the left as much as possible, while ensuring there remains enough space for the thick edges and attempting to displace each obstacle by at most δ. For all $1 \leq i \leq n_o$ and $\delta \geq 0$, we define the displacement function $X_\delta[i]$ as

$$X_\delta[i] = \begin{cases} x_i - \delta & \text{if } i = 1 \\ \max(x_i - \delta, X_\delta[i-1] + \tau(E_{i-1,i})) & \text{otherwise.} \end{cases} \tag{4.3}$$

Given some displacement $\delta \geq 0$, we can compute the new obstacle positions by setting the new position of each obstacle ω_i equal to $X_\delta[i]$ for all $1 \leq i \leq n_o$. If δ is chosen too small such that no valid solution with maximum displacement δ exists, the solution constructed by X_δ does not provide any guarantee about the maximum displacement applied to the obstacles. We argue that displacing the obstacles using X_δ with $\delta = \delta_{min}$ produces a valid and optimal solution to the problem with a maximum displacement of δ_{min}, which we will prove in Section 4.2.1.

4.2.1 Correctness

The proof of correctness for the algorithm consists of two key parts: proving that X_δ produces a valid solution, and proving that $X_{\delta_{min}}$ produces an optimal solution with a maximum displacement of δ_{min}. Let us first prove two auxiliary lemmas that will later help us prove the main results.

Auxiliary lemmas

Lemma 1. $\delta_{min} = \max_{1 \le j \le i \le n_o} \frac{\tau(E_{j,i}) - (x_i - x_j)}{2}$.

Proof. We prove that $D[i] = \max_{1 \le j \le i} (\tau(E_{j,i}) - (x_i - x_j))$ for all $1 \le i \le n_o$ by induction on i, thereby proving the statement. For $i = 1$, we have that

$$\max_{1 \le j \le 1} (\tau(E_{j,1}) - (x_1 - x_j)) = \tau(E_{1,1}) - (x_1 - x_1) = 0 = D[1].$$

As induction hypothesis (IH), assume that $D[k] = \max_{1 \le j \le k} (\tau(E_{j,k}) - (x_k - x_j))$ holds for some $1 \le k < n_o$. Then, we have that

$$\begin{aligned} D[k+1] &= \max(0, D[k] + \tau(E_{k,k+1}) - (x_{k+1} - x_k)) \\ &= \max(0, \max_{1 \le j \le k} (\tau(E_{j,k}) - (x_k - x_j)) + \tau(E_{k,k+1}) - (x_{k+1} - x_k)) && \text{by IH} \\ &= \max(0, \max_{1 \le j \le k} (\tau(E_{j,k+1}) - (x_{k+1} - x_j))) \\ &= \max_{1 \le j \le k+1} (\tau(E_{j,k+1}) - (x_{k+1} - x_j)). \end{aligned}$$

Hence, $D[i] = \max_{1 \le j \le i} (\tau(E_{j,i}) - (x_i - x_j))$ for all $1 \le i \le n_o$ and thus $\delta_{min} = \max_{1 \le j \le i \le n_o} \frac{\tau(E_{j,i}) - (x_i - x_j)}{2}$. □

Lemma 2. $X_\delta[i] = \max_{1 \le j \le i} (x_j - \delta + \tau(E_{j,i}))$ *for all* $1 \le i \le n_o$ *and* $\delta \ge 0$.

Proof. Let $\delta \ge 0$. We prove the statement by induction on i. For $i = 1$, we have that

$$\max_{1 \le j \le 1} (x_j - \delta + \tau(E_{j,1})) = x_1 - \delta + \tau(E_{1,1}) = x_1 - \delta = X_\delta[1].$$

As induction hypothesis (IH), assume that $X_\delta[k] = \max_{1 \le j \le k} (x_j - \delta + \tau(E_{j,k}))$ holds for some $1 \le k < n_o$. Then, we have that

$$\begin{aligned} X_\delta[k+1] &= \max(x_{k+1} - \delta, X_\delta[k] + \tau(E_{k,k+1})) \\ &= \max(x_{k+1} - \delta, \max_{1 \le j \le k} (x_j - \delta + \tau(E_{j,k})) + \tau(E_{k,k+1})) && \text{by IH} \\ &= \max(x_{k+1} - \delta, \max_{1 \le j \le k} (x_j - \delta + \tau(E_{j,k+1}))) \\ &= \max_{1 \le j \le k+1} (x_j - \delta + \tau(E_{j,k+1})). \end{aligned}$$

Hence, $X_\delta[i] = \max_{1 \le j \le i} (x_j - \delta + \tau(E_{j,i}))$ for all $1 \le i \le n_o$ and $\delta \ge 0$. □

Correctness of X_δ

To prove correctness of X_δ, we need to prove that after displacing each obstacle ω_i to position $X_\delta[i]$ for all $1 \le i \le n_o$ and some displacement $\delta \ge 0$, there is enough space to draw the thick edges such that they are homotopic to the input edges. In other words, the distance between each pair of adjacent obstacles must be at least their separating edge thickness, and the obstacles must maintain their original x-order. This is reflected by the following lemma.

Lemma 3 (Correctness of X_δ)**.** $X_\delta[i+1] - X_\delta[i] \ge \tau(E_{i,i+1})$ *for all* $1 \le i < n_o$ *and* $\delta \ge 0$.

Proof. Let $\delta \ge 0$. For all $1 \le i < n_o$, we have that

$$\begin{aligned} X_\delta[i+1] - X_\delta[i] &= \max(x_{i+1} - \delta, X_\delta[i] + \tau(E_{i,i+1})) - X_\delta[i] \\ &\ge X_\delta[i] + \tau(E_{i,i+1}) - X_\delta[i] \\ &= \tau(E_{i,i+1}). \end{aligned}$$

Hence, $X_\delta[i+1] - X_\delta[i] \ge \tau(E_{i,i+1})$ for all $1 \le i < n_o$ and $\delta \ge 0$. □

Optimality of $X_{\delta_{min}}$

To prove that $X_{\delta_{min}}$ produces an optimal solution, we need to prove that $X_{\delta_{min}}$ displaces each obstacle by at most δ_{min}, and that δ_{min} is the smallest maximum displacement of any valid solution to the problem.

Lemma 4 (Maximum displacement of δ_{min})**.** *$x_i - \delta_{min} \leq X_{\delta_{min}}[i] \leq x_i + \delta_{min}$ for all $1 \leq i \leq n_o$.*

Proof. Clearly, we have that $x_i - \delta_{min} \leq X_{\delta_{min}}[i]$ by definition of $X_{\delta_{min}}$, so it remains to prove that $X_{\delta_{min}}[i] \leq x_i + \delta_{min}$. For all $1 \leq i \leq n_o$, we have that

$$\begin{aligned}
X_{\delta_{min}}[i] - x_i &= \max_{1 \leq j \leq i} (x_j - \delta_{min} + \tau(E_{j,i})) - x_i && \text{by Lemma 2} \\
&\leq \max_{1 \leq j \leq i} \left(x_j - \frac{\tau(E_{j,i}) - (x_i - x_j)}{2} + \tau(E_{j,i}) \right) - x_i && \text{by Lemma 1} \\
&= \max_{1 \leq j \leq i} \frac{\tau(E_{j,i}) - (x_i - x_j)}{2} \\
&\leq \delta_{min} && \text{by Lemma 1.}
\end{aligned}$$

Hence, $x_i - \delta_{min} \leq X_{\delta_{min}}[i] \leq x_i + \delta_{min}$ for all $1 \leq i \leq n_o$. □

Lemma 5 (Minimality of δ_{min})**.** *δ_{min} is the smallest maximum displacement of any valid solution to the problem.*

Proof. Choose $1 \leq i' \leq n_o$ and $1 \leq j' \leq i'$ such that, by Lemma 1,

$$\delta_{min} = \max_{1 \leq j \leq i \leq n_o} \frac{\tau(E_{j,i}) - (x_i - x_j)}{2} = \frac{\tau(E_{j',i'}) - (x_{i'} - x_{j'})}{2}.$$

We can rewrite this to $x_{i'} - x_{j'} + 2\delta_{min} = \tau(E_{j',i'})$. Suppose, towards a contradiction, that there exists a maximum displacement $\delta' \geq 0$ such that $\delta' < \delta_{min}$ and there exists a valid solution with maximum displacement δ'. Then, the obstacles $\omega_{j'}$ and $\omega_{i'}$ are maximally separated by displacing $\omega_{j'}$ by δ' to the left, and $\omega_{i'}$ by δ' to the right. However, now we have that

$$(x_{i'} + \delta') - (x_{j'} - \delta') = x_{i'} - x_{j'} + 2\delta' < \tau(E_{j',i'}).$$

Hence, after maximally separating $\omega_{j'}$ and $\omega_{i'}$ using the maximum displacement δ', there is still not enough space to draw the thick edges between the two obstacles, which contradicts that the solution is valid. Therefore, δ' cannot exist and δ_{min} must be the smallest maximum displacement of any valid solution to the problem. □

4.2.2 Running time

The algorithm consists of determining δ_{min}, the smallest maximum displacement of any valid solution to the problem, and then using δ_{min} to optimally displace the obstacles. Computing the value of δ_{min} requires the computation of $D[i]$ for all $1 \leq i \leq n_o$. Subsequently, the new position of each obstacle ω_i is found by computing $X_{\delta_{min}}[i]$ for all $1 \leq i \leq n_o$. We assume that the values of $\tau(E_{i-1,i})$ for all $1 < i \leq n_o$ are known in advance, which can trivially be computed using a linear scan over the obstacles and edges in $\mathcal{O}(n_o + |E|)$ time. Hence, both recurrences involve solving n_o subproblems, and each subproblem can be solved in constant time. Therefore, the total running time of the algorithm is linear in the number of obstacles.

Theorem 1. *The one-dimensional MDGE problem can be solved in $\mathcal{O}(n_o)$ time.*

Chapter 5

NP-hardness

In this chapter, we show that, if the goal is to minimize the maximum obstacle displacement, a modified version of the MDGE problem is NP-hard. In particular, we allow the thick edges to overlap, while they must still be disjoint from the obstacles and homotopic to the input edges. Although we do not directly address the complexity of the actual MDGE problem, already a slight modification makes it NP-hard, suggesting that the MDGE problem is likely NP-hard as well.

We prove NP-hardness by reduction from a special version of 3-SAT, which is a well-known NP-complete decision problem. A 3-SAT instance consists of a boolean formula $\Phi = (X, C)$ in conjunctive normal form, where X is the set of variables and C is a set of clauses on these variables consisting of two or three literals. The goal is to determine whether there exists an assignment of truth values to the variables in X such that all clauses in C are satisfied. A clause is called *monotone* if it consists of only positive literals (a *positive clause*) or only negative literals (a *negative clause*). Even when restricted to monotone clauses, 3-SAT remains NP-complete [33]. Another variant of the 3-SAT problem requires that the incidence graph of the variables and (not necessarily monotone) clauses, that is, the graph $G_\Phi = (X \cup C, \{(x, c) \in X \times C \mid x \in c \vee \overline{x} \in c\})$, is planar. This version was proven to be NP-complete by Lichtenstein [59]. As shown by Knuth and Raghunathan [51], G_Φ always allows a *rectilinear representation* in which the variables and clauses are represented by axis-aligned rectangles (see Figure 5.1). In this representation, the rectangles representing the variables are placed on a horizontal line and are connected to their respective clause rectangles by non-crossing vertical line segments.

The *planar monotone 3-SAT problem* combines the monotonicity and planarity constraints such that the 3-SAT instance only contains monotone clauses and has a planar incidence graph. This variant was introduced and proven to be NP-complete by De Berg and Khosravi [8]. They show how to construct a *monotone rectilinear representation* of a planar monotone 3-SAT instance, which is a rectilinear representation with all positive clauses drawn above the variables and all negative clauses drawn below the variables.

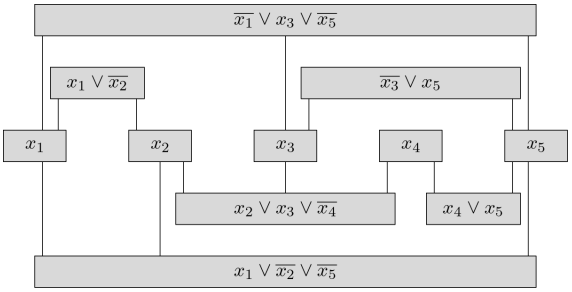


Figure 5.1: A rectilinear representation of the incidence graph G_Φ of a planar 3-SAT instance Φ, with satisfying assignment $x_1 = x_2 = x_4 = \textit{true}$ and $x_3 = x_5 = \textit{false}$.

To prove that the MDGE problem with allowed overlapping thick edges is NP-hard, we show how to reduce a monotone rectilinear representation of a planar monotone 3-SAT instance to an instance of the MDGE problem. To this end, we introduce small instances of the MDGE problem, so-called *gadgets*, to represent the various components of the rectilinear representation: variables, clauses and connectors (the segments connecting the variables to their respective clauses). These gadgets are then assembled to construct an instance of the MDGE problem that simulates the behavior of the corresponding planar monotone 3-SAT instance, thereby establishing the reduction.

Gadgets

Consider the construction in Figure 5.2(a). Let $v_1, v_2, v_3, v_4 \in V$ denote, from top to bottom, the four vertices shown in black and let $o \in O$ denote the obstacle shown in gray. Note that in our gadgets, we allow vertices that are not incident to any edge, which are used to restrict the movement of the obstacles. The points are evenly spaced such that $d(v_1, v_2) = d(v_2, o) = d(v_3, o) = d(v_3, v_4) = 1$. Furthermore, let $e = \{v_1, v_4\}$ denote the blue edge with thickness $\tau(e) = \sqrt{2}$. To draw the edge with its desired thickness and homotopy disjoint from the obstacle and vertices, we must create extra space between v_2 and o and between v_3 and o. To be precise, the obstacle must lie outside of the circles with radius $\sqrt{2}$ centered at v_2 and v_3 shown in the figure. As a result, there are exactly two ways to solve the instance using minimum displacement: by displacing o by one to the left (see Figure 5.2(b)) or by one to the right (see Figure 5.2(c)). It is easy to see that the resulting thick edges are homotopic to the input edge. This construction is used as connector gadget, whose solution represents the truth value of a certain literal. In particular, if the obstacle is displaced by one to the left, the corresponding literal is evaluated to *true*, while displacing the obstacle by one to the right represents a *false* evaluation of the literal.

By placing multiple connector gadgets next to each other with a separation of 2, as shown in Figure 5.2(d), we can propagate the truth value of a literal through the instance along a so-called *connector branch*. If the blue connector gadget in the figure evaluates to *true*, the red connector gadget may evaluate to either *true* or *false*. However, since each connector branch will end at a clause gadget whose corresponding clause contains the respective literal, the incentive is to evaluate the literal to *true*, thereby stimulating the solution shown in Figure 5.2(e). Alternatively, if the blue connector gadget evaluates to *false*, the red connector gadget cannot evaluate to *true*, as this would place its obstacle on top of the previous obstacle. Therefore, all subsequent connector gadgets are forced to evaluate to *false* as well, as shown in Figure 5.2(f).

To be able to connect literals to multiple clauses, we need additional connector gadgets to propagate truth values along bends and split branches into multiple directions. The construction in Figure 5.3(a) shows how a branch is rotated by 90 degrees through the addition of an extra obstacle. If the incoming blue branch evaluates to *true*, the extra obstacle can be displaced upwards by one, allowing the outgoing red branch to be evaluated to *true* as well (see Figure 5.3(b)). If however the incoming branch evaluates to *false*, the extra obstacle must be displaced by one to the right, thereby forcing the outgoing branch to be evaluated to *false* (see Figure 5.3(c)). Similarly, we can split a branch into two new branches using the splitter gadget shown in Figure 5.3(d). In

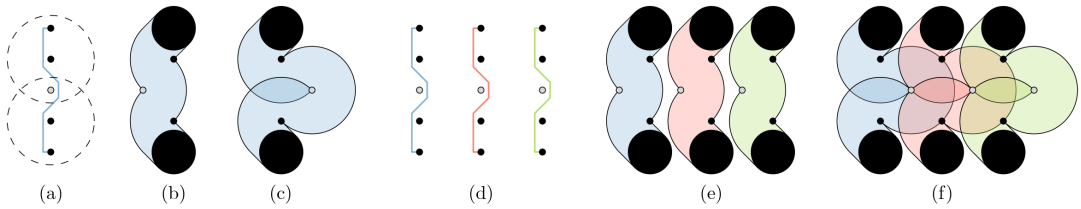


Figure 5.2: Basic connector gadgets. (a) A single connector gadget. (b) Displacing the obstacle by one to the left represents a *true* literal. (c) Displacing the obstacle by one to the right represents a *false* literal. (d) A branch of connector gadgets can be used to propagate the truth value of a literal. (e) Propagation of a *true* literal. (f) Propagation of a *false* literal.

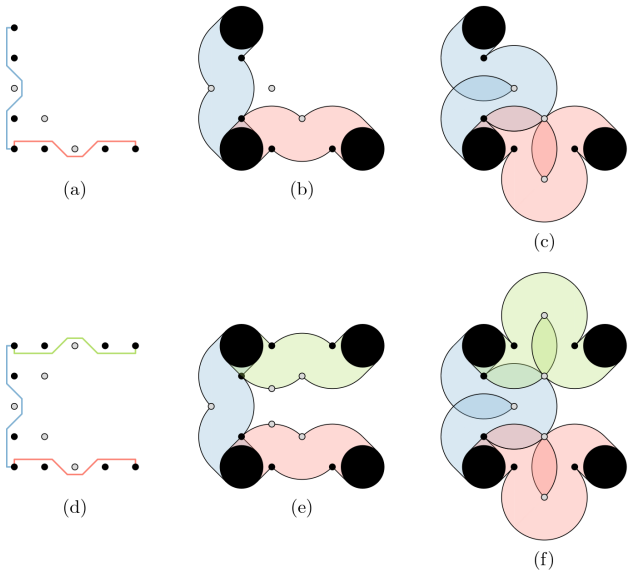


Figure 5.3: Additional connector gadgets. (a) A bend gadget. (b, e) Propagation of a *true* literal through the bend/splitter gadget. (c, f) Propagation of a *false* literal through the bend/splitter gadget. (d) A splitter gadget.

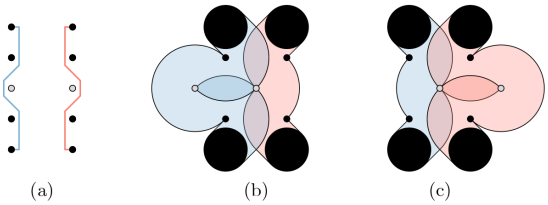


Figure 5.4: Variable gadgets. (a) A variable gadget. (b) Displacing both obstacles by one to the left represents the value *true*. (c) Displacing both obstacles by one to the right represents the value *false*.

this case two extra obstacles are introduced, splitting the incoming blue branch into two opposite branches. The propagation of the incoming truth value follows the same principle as for bends (see Figure 5.3(e) and Figure 5.3(f)). Note that in both the bend and splitter gadget, the incoming connector gadget intersects each outgoing connector gadget in exactly one vertex. This issue is solved by simply replacing the intersecting vertices by a single vertex incident on both edges.

Our variable gadget consists of two connector gadgets of which one is inverted, as shown in Figure 5.4(a). While a single connector gadget is sufficient to force a binary choice in obstacle displacement, this construction is used to make the branches on either side of the variable gadget symmetric. Displacing the obstacle of the right connector gadget by one to the left represents the value *true* (see Figure 5.4(b)), while displacing the obstacle of the left connector gadget by one to the right represents the value *false* (see Figure 5.4(c)). Clearly only one of these cases is possible. The right side of each variable gadget is connected to a branch representing the positive literal, while its left side is connected to a branch representing the negative literal. Hence, if the variable evaluates to *true*, the branch corresponding to the positive literal propagates the truth value *true*, while the branch corresponding to the negative literal propagates the truth value *false*. Conversely, if the variable evaluates to *false*, the branch corresponding to the negative literal propagates the truth value *true*, while the branch corresponding to the positive literal propagates the truth value *false*. Each branch must be connected to all clauses containing the respective literal. Since a variable gadget consists of two connector gadgets itself, it is trivial to construct the branches on

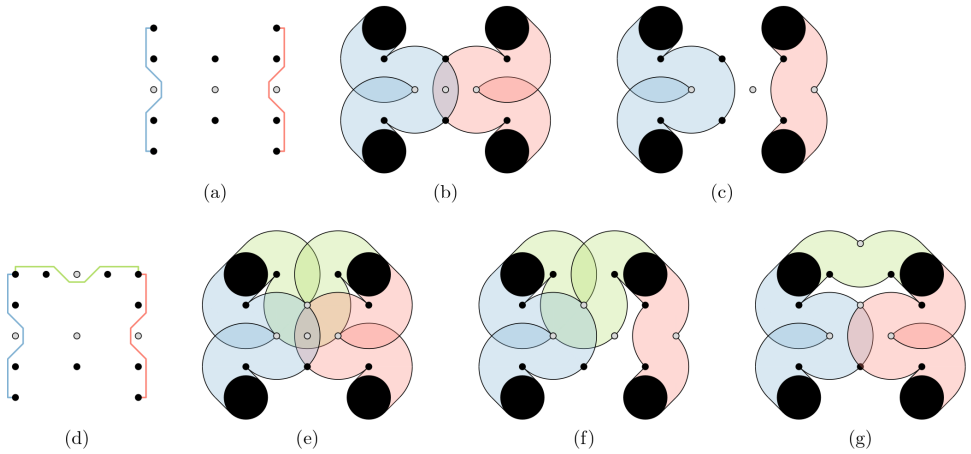


Figure 5.5: Clause gadgets. (a) A two-literal clause gadget. (b, e) If all incoming branches evaluate to *false*, the corresponding clause is unsatisfiable. (c, f, g) If at least one incoming branch evaluates to *true*, the corresponding clause is satisfiable. (d) A three-literal clause gadget.

either side of the variable gadget using the connector gadgets given in Figure 5.2 and Figure 5.3.

In our clause gadgets, two or three branches arrive representing the literals of the clause. The constructions shown in Figure 5.5(a) and Figure 5.5(d) are the gadgets used for clauses consisting of respectively two and three literals. An extra obstacle is added, along with one or two extra vertices to limit the obstacle's movement. If all incoming branches evaluate to *false*, there does not exist a solution in which the extra obstacle is displaced by at most one (see Figure 5.5(b) and Figure 5.5(e)). This represents that the corresponding clause is unsatisfiable. If however at least one incoming branch evaluates to *true*, the corresponding clause is satisfiable as the extra obstacle can be displaced by at most one towards a branch representing a *true* literal (see Figure 5.5(c), Figure 5.5(f) and Figure 5.5(g)).

Reduction

Consider a monotone rectilinear representation of a planar monotone 3-SAT instance. We show how to reduce it to an instance of the MDGE problem using our gadgets (see Figure 5.6 for an example). First we rotate the representation by 90 degrees such that all variables lie on a vertical line and the negative and positive clauses are respectively on the left and right side of the variables. For each variable, we replace its outgoing segments by a single segment for the positive clauses and a single segment for the negative clauses. Since the clauses on either side of the variable are either all positive or all negative and the instance is planar, the connections can be trivially reconstructed by branching the segments towards the respective clauses via rectilinear non-crossing paths. We route these paths such that for each clause, the incoming segments are attached to different sides of the rectangle, and, for two-literal clauses, are attached to opposite sides. Furthermore, we ensure that we only use splitters compatible with our splitter gadget, that is, segments may only split into two opposite branches.

We can represent the variables and clauses using the gadgets described above. Due to the altered rectilinear representation, the connector gadgets can naturally be used to form the connections from the variable gadgets to their respective clause gadgets, completing the reduction.

Theorem 2. *The MDGE problem with allowed overlapping thick edges is NP-hard.*

Proof. We showed how an instance of the planar monotone 3-SAT problem can be reduced to an instance of the MDGE problem. It is easy to see that the reduction can be computed in polynomial time. To prove NP-hardness, it remains to prove that a planar monotone 3-SAT instance Φ is satisfiable if and only if the associated MDGE instance can be solved with overlapping thick edges and a maximum displacement of one.

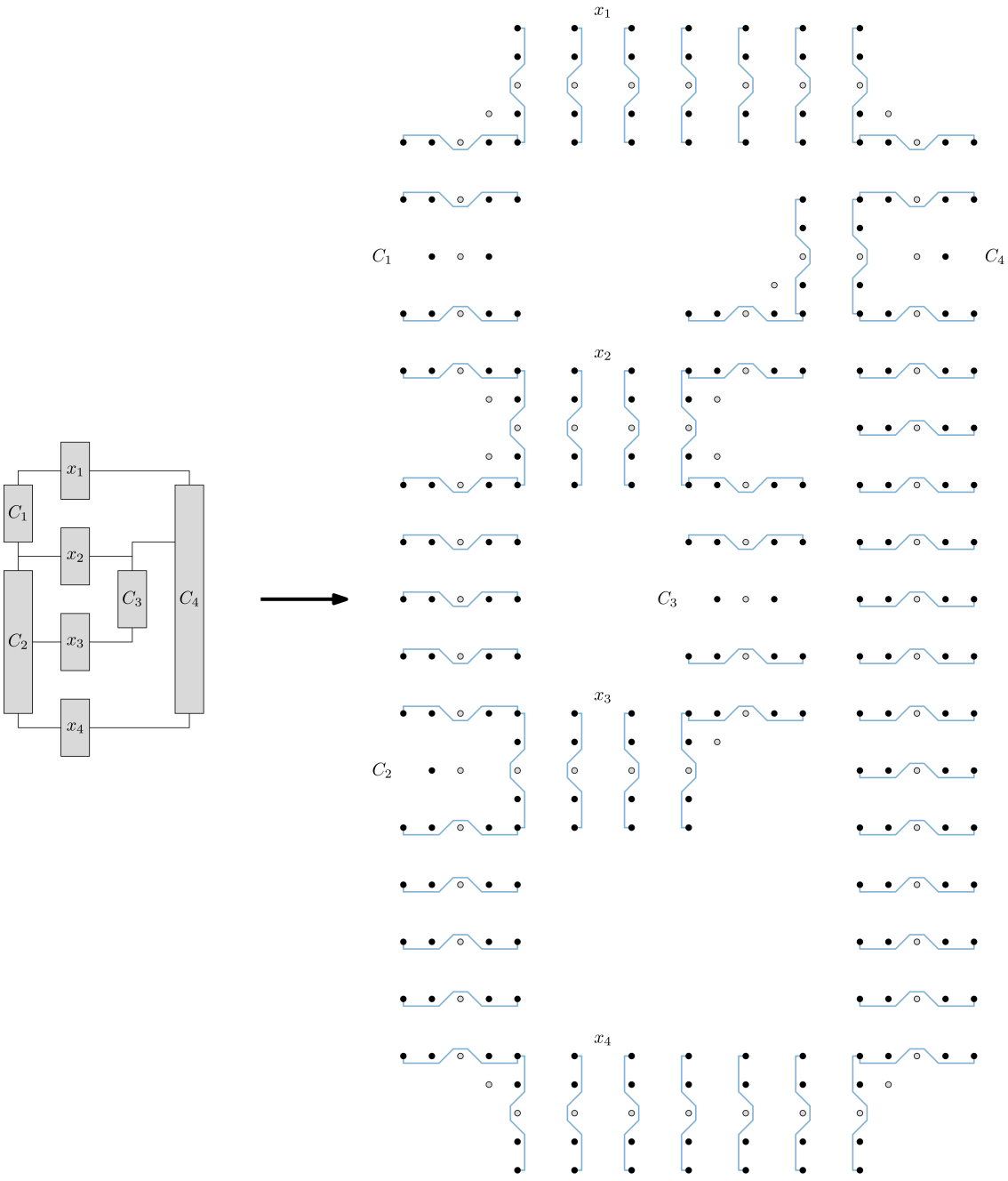


Figure 5.6: A reduction from the planar monotone 3-SAT instance $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$ with $C_1 = (\overline{x_1} \vee \overline{x_2})$, $C_2 = (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4})$, $C_3 = (x_2 \vee x_3)$ and $C_4 = (x_1 \vee x_2 \vee x_4)$.

Consider a satisfying truth assignment for Φ. The obstacles of each variable gadget can be displaced by one such that they represent the truth value of the corresponding variable. By construction, the connector gadgets forming the respective branches on either side of the variable gadget can propagate this truth value by displacing their obstacles by one as well. The branches represent the positive and negative literal of the variable and are connected to all clause gadgets whose clauses contain the corresponding literal. Therefore, since each clause in Φ is satisfied, at least one of the incoming branches of each clause gadget evaluates to *true*. This allows the extra obstacles of the clause gadgets to be displaced by at most one to obtain a valid solution. The figures depict all relevant configurations of the connector, variable and clause gadgets after displacing the obstacles as described. From this it is easy to see that the resulting construction forms a valid solution to the MDGE problem with overlapping thick edges.

Conversely, if all obstacles in the MDGE instance corresponding to Φ can be displaced by at most one such that we obtain a valid solution with overlapping thick edges, each variable can be assigned a truth value based on the positions of the obstacles of the corresponding variable gadget. If one of the obstacles is displaced 'inwards', the variable gadget is associated with a single truth value and evaluates the corresponding literal to *true*. If both obstacles are displaced 'outwards', both literals evaluate to *false* and thus the corresponding variable may be assigned either *true* or *false*. By construction, each clause gadget is connected to a branch representing a *true* literal. Therefore, the assignment of truth values to variables satisfies all clauses and hence forms a satisfying truth assignment for Φ. □

Chapter 6

Solving the MDGE problem

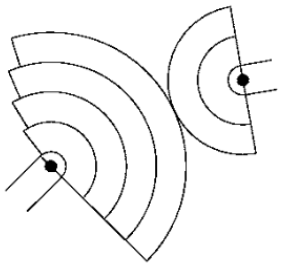
In this chapter, we develop a practical algorithm for the MDGE problem. We start by investigating methods to check whether a solution is valid. To this end, we first study the algorithm by Duncan et al. [23] for growing thick homotopic edges. We then observe that we can model the MDGE problem as a constrained optimization problem, with constraints specifying the minimum required separation between each pair of points. By only imposing constraints on the edges of a Delaunay triangulation on the vertices and obstacles, we obtain a 1.998-approximation of the minimum-separation constraints, while reducing the number of constraints from quadratic to linear. The resulting optimization problem is efficiently solved using a linear program with additional constraints on the orthogonal order of the points. This linear program is combined with the growing algorithm for computing thick homotopic edges to develop a practical algorithm.

6.1 Growing thick edges

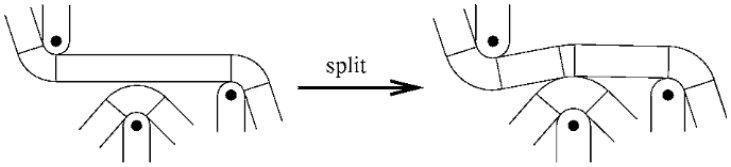
We can compute the thick homotopic edges using the approach by Duncan et al. [23]. Given a set of crossing-free paths, their algorithm grows thick disjoint shortest paths with maximum thickness that are homotopic to the input paths. The algorithm grows the paths simultaneously in thickness over time, at a speed proportional to the individual weight of each path. Throughout this growth process, the sum of the lengths of all paths remains as small as possible, and the homotopy among paths is preserved. We first describe the growing algorithm given in [23] and then show how to adapt it for use in our algorithm for the MDGE problem.

The paper divides the structure of a thick edge into terminal elbows (at the endpoints), non-terminal elbows (at bends) and straights. An elbow is a part of the annulus around a vertex or obstacle, while a straight is simply a straight segment connecting two elbows. To prevent the edges from overlapping the obstacles during the growth process of the algorithm, each obstacle is also assigned a terminal elbow with weight zero. An elbow is associated with the point it bends around, while a straight is associated with the two points associated to the elbows it connects. During the growth process, the algorithm maintains a priority queue of events, with the key being the time at which an event occurs. There are three types of events: split, merge and stop events (see Figure 6.1). A split event splits a straight into a straight-elbow-straight sequence when it is hit by an elbow. A merge event does the opposite by merging a straight-elbow-straight sequence into a single straight if the elbow is straightened. A stop event terminates the growth process, which happens when two elbows collide.

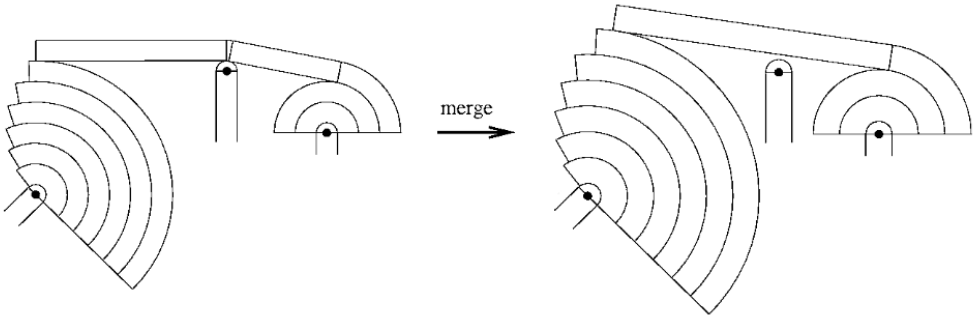
To maintain the state of the growth process more efficiently, the paper introduces a compact routing structure that bundles straights and elbows together, respectively (see Figure 6.2). A straight bundle groups all touching parallel straights that are associated to the same points, while an elbow bundle groups all touching parallel elbows that are associated to the same point and share the same straight bundles on both ends. A terminal elbow forms its own bundle. The split, merge and stop events are simply applied on all segments within the corresponding bundle.



(a) A stop event.



(b) A split event.



(c) A merge event.

Figure 6.1: Illustrations of the different types of events during the growth process of the growing algorithm. Adapted from [23].

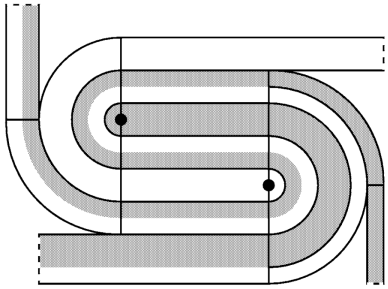


Figure 6.2: An illustration of the straight and elbow bundles maintained by the compact routing structure, where the bundle boundaries are shown in bold. The two half-disks at the vertices are terminal elbows. Adapted from [23].

After performing a split or merge event, the compact routing structure is updated by merging or splitting parallel bundles when necessary. If a stop event is triggered, the individual paths are retrieved by unzipping the bundles. This is done by repeatedly 'tearing' the top segment off from each straight bundle until they all consist of exactly one segment, after which each elbow bundle is also left with a single segment by definition.

We adapt the growing algorithm as follows. Instead of maximizing the thickness of the edges, we add a stop event to the growing process to signal termination if the edges have reached their required thickness. To extend the growing algorithm to arbitrary graphs, we want to draw each vertex as the smallest disk such that its incident edges do not overlap outside that disk. Since the edge thicknesses in the MDGE problem are predefined, we can precompute the required vertex sizes. The growing algorithm runs in $\mathcal{O}(m^3 + k)$ time and uses $\mathcal{O}(m + k)$ space, where m is the number of edges in E and k is the maximum of the input and output complexities of all paths.

To check whether a given solution to the MDGE problem is valid, we can simply run the growing algorithm to grow thick homotopic edges. If the edges have reached their required thickness, the solution is valid. Otherwise, the edges could not be grown any further while remaining interior disjoint and thus no solution is possible. While this does provide a method to check for validity, it is not practical due to the algorithm's complexity. At a first glance, it may seem possible to dynamically displace the obstacles by letting the thick edges 'push' them away during the growing process. However, this leads to many choices in how to displace the obstacles and it is not trivial to see what would be a good displacement strategy.

6.2 Constrained optimization

To obtain a more practical validity check for the MDGE problem, we observe that it can be modeled as a constrained optimization problem, with constraints specifying the minimum required separation between pairs of obstacles. Consider two arbitrary points $p, q \in V \cup O$ and an edge $e_i \in E$ with shortest homotopic path π_i. We say that π_i *separates* p and q if and only if π_i intersects line segment $\overline{pq}$. Let $c^i_{sep}(p, q)$ denote the number of times path π_i separates p and q, that is, the number of intersections between π_i and $\overline{pq}$. We use $\tau_{sep}(p, q)$ to denote the total edge thickness separating p and q, that is, $\tau_{sep}(p, q) = r(p) + r(q) + \sum_{1 \le i \le m} c^i_{sep}(p, q) \cdot \tau(e_i)$, where $r(p)$ is the radius of vertex p, which is 0 if p is an obstacle. We argue that the graph can be drawn with thick interior disjoint edges that are homotopic to the input edges if and only if the distance between each pair of points is at least their separating edge thickness, which we prove using the following lemma.

Lemma 6. *Graph G admits a planar thick embedding if and only if $d(p, q) \ge \tau_{sep}(p, q)$ for all points $p, q \in V \cup O$.*

Proof. Consider a planar thick embedding of G in which each edge e_i is embedded as a thick path π'_i homotopic to its input path π_i. Let $p, q \in V \cup O$ be two arbitrary points. Suppose that π_i separates p and q. Since π_i is shortest and homotopic to π'_i, the thick path π'_i must also separate p and q. Hence, as the thick paths in the planar thick embedding are interior disjoint, we must have that $d(p, q) \ge \tau_{sep}(p, q)$.

Now let $d(p, q) \ge \tau_{sep}(p, q)$ for all points $p, q \in V \cup O$ and suppose, towards a contradiction, that G does not admit a planar thick embedding. Consider the algorithm for growing thick homotopic edges by Duncan et al. [23]. By Lemma 1 of [23], the algorithm terminates when the edges cannot be grown any further while remaining interior disjoint. This happens when the first stop event is triggered, in which case two elbow bundles with distinct center points $u, v \in V \cup O$ collide. If the edges have reached their required thickness, we clearly have a planar thick embedding, so suppose that the edges did not reach their required thickness but have maximum thickness proportional to their weight. Clearly, the line segment $\overline{uv}$ intersects all thick elbow bundles associated with u and v. Throughout the growth process, the paths remain shortest and homotopic to the input paths. Therefore, $\overline{uv}$ must also intersect all input paths, which contradicts that $d(u, v) \ge \tau_{sep}(u, v)$. □

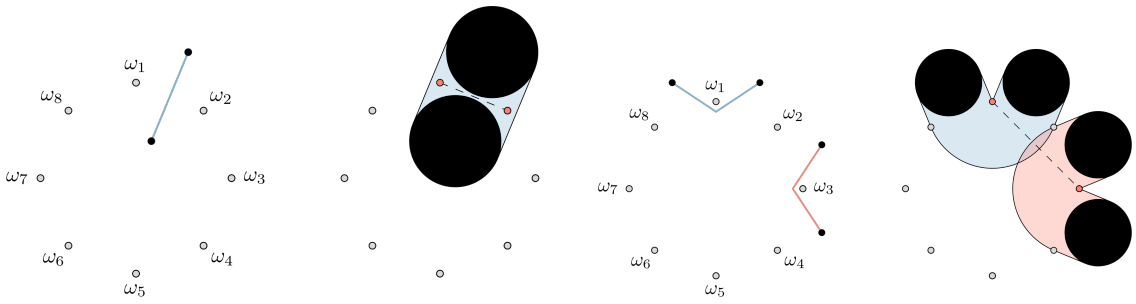
Let $O = \{\omega_1, \ldots, \omega_{n_o}\}$ be the set of obstacles and let ω'_i denote the new position of obstacle ω_i for all $1 \le i \le n_o$. Following Lemma 6, we can encode the obstacle displacement problem as the following constrained optimization problem:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n_o} d(\omega_i, \omega'_i) \\
\text{subject to} \quad & d(\omega'_i, \omega'_j) \ge \tau_{sep}(\omega_i, \omega_j) && \text{for all } 1 \le i < j \le n_o, \\
& d(\omega'_i, v) \ge \tau_{sep}(\omega_i, v) && \text{for all } 1 \le i \le n_o \text{ and } v \in V, \\
& d(u, v) \ge \tau_{sep}(u, v) && \text{for all } u, v \in V.
\end{aligned}
$$

The number of constraints of the above optimization problem is quadratic in the number of vertices and obstacles, which is typically not desired. Clearly, some constraints may not be necessary as they are indirectly enforced by other constraints. However, it is not trivial to determine which constraints can be safely omitted while still ensuring a valid solution. This is demonstrated by the following lemma.

Lemma 7. *There exists a set of obstacles O such that for each pair of distinct obstacles, there exists a graph G and an embedding for it such that the minimum-separation constraint on that pair is necessary.*

Proof. Let n_o be a multiple of 4 and let the obstacles in O be evenly distributed on the unit circle such that $d(\omega_i, \omega_{i+1}) = d(\omega_{n_o}, \omega_1)$ for all $1 \leq i < n_o$. Due to our symmetric configuration of obstacles, it suffices to prove the statement for each pair of obstacles (ω_1, ω_k) with $2 \leq k \leq n_o/2+1$. To prove that, due to our construction of graph G, a minimum-separation constraint on the pair of obstacles (ω_1, ω_k) is necessary, we show that it is the only pair of obstacles for which the minimum-separation constraint is violated for G. We construct graph G as follows (see Figure 6.3 for an example with $n_o = 8$). If $k = 2$, we simply route an edge between ω_1 and ω_2 towards the center of the circle with thickness slightly larger than $d(\omega_1, \omega_2)$. Clearly, only the minimum-separation constraint on the pair (ω_1, ω_2) is violated. Now let $3 \leq k \leq n_o + 1$. For all $1 \leq i < k - 1$, we route an edge between ω_i and ω_{i+1} such that it 'exits' the circle of obstacles on the other side. The combined thickness of the edges between ω_1 and ω_{k-1} is maximized while ensuring that the constraint on each pair (ω_i, ω_j) is not violated for all $1 \leq i < j < k$, that is, $\tau_{sep}(\omega_i, \omega_j) = d(\omega_i, \omega_j)$. Additionally, we route an edge between ω_{k-1} and ω_k with thickness $d(\omega_{k-1}, \omega_k)$. However, we now have that $\tau_{sep}(\omega_1, \omega_k) = \tau_{sep}(\omega_1, \omega_{k-1}) + \tau_{sep}(\omega_{k-1}, \omega_k) = d(\omega_1, \omega_{k-1}) + d(\omega_{k-1}, \omega_k) > d(\omega_1, \omega_k)$. Hence, only the minimum-separation constraint on the pair (ω_1, ω_k) is violated. □



(a) Case 1: (ω_1, ω_2).

(b) Case 2: (ω_1, ω_3).

(c) Case 3: (ω_1, ω_4).

(d) Case 4: (ω_1, ω_5).

Figure 6.3: Illustrations of the graph constructions for the proof of Lemma 7 with $n_o = 8$. There are four cases we need to consider, the other cases are symmetric. In each case, the pair of red obstacles is the only pair for which the minimum-separation constraint is violated.

6.3 Delaunay approximation

While it is possible to enforce minimum-separation constraints on all pairs of points, this yields a quadratic number of constraints, which is typically not desired. We propose to only introduce constraints on the edges of a Delaunay triangulation on the vertices and obstacles, which leads to a linear number of constraints at the cost of increasing the minimum required separation of each constraint by a certain factor. Let D be a Delaunay triangulation on the vertices in V and the obstacles in O. We define the *minimum separation factor* of D as the smallest value $\sigma > 1$ such that, if $|uv| \geq \sigma \cdot \tau_{sep}(u, v)$ for each edge (u, v) in D, we have that $d(p, q) \geq \tau_{sep}(p, q)$ for all $p, q \in V \cup O$. In other words, the minimum separation factor is the smallest factor by which we can increase the minimum required separation of the constraints, such that only imposing minimum-separation constraints on the Delaunay edges correctly separates all pairs of points. Figure 6.4 shows an example construction where the minimum separation factor approaches $\pi/2$, which is also a lower bound on the stretch factor of a Delaunay triangulation as shown by Chew [16]. Therefore, the stretch factor seems to be a lower bound for the minimum separation factor. We use a result by Xia [100] on the upper bound on the stretch factor of a chain of disks to obtain an upper bound of 1.998 on the minimum separation factor of D. Let us first introduce some important concepts and notation used in the paper (see Figure 6.5).

Let $\mathcal{C} = (C_1, C_2, \ldots, C_n)$ be a sequence of n distinct disks in the plane such that every two consecutive disks C_i and C_{i+1} intersect for $1 \leq i < n$, but neither disk contains the other. Furthermore, for each disk C_i with $1 < i < n$, the two arcs on its boundary that intersect C_{i-1} and C_{i+1} do not overlap, but may share an endpoint. We say that $\mathcal{C}$ is a *chain*, and two points p and q are called *terminals* of $\mathcal{C}$ if p lies on the boundary of C_1 outside of C_2 and q lies on the boundary of C_n outside of C_{n-1}. Define the *centered polyline* between p and q as the polyline $pc_1 \ldots c_n q$, where c_i denotes the center of disk C_i for all $1 \leq i \leq n$. For $1 \leq i < n$, the boundaries of every two consecutive disks C_i and C_{i+1} intersect in two points a_i and b_i, where $a_i = b_i$ if the disks are tangent. Without loss of generality, we assume that all a_i's are on one side of the centered polyline between p and q and all b_i's are on the other side, where they are both on the centered polyline if $a_i = b_i$. For notational convenience, let $a_0 = b_0 = p$ and $a_n = b_n = q$. For $1 \leq i \leq n$, let A_i and B_i denote the arcs on the boundary of disk C_i from a_{i-1} to a_i and from b_{i-1} to b_i, respectively.

Let $D_{\mathcal{C}}(p, q)$ be the shortest polyline from p to q through $\mathcal{C}$ that traverses the sequence of disks in order. That is, $D_{\mathcal{C}}(p, q)$ intersects line segments $\overline{a_1 b_1}, \ldots, \overline{a_{n-1} b_{n-1}}$ in that order. The paper defines the shortest path between p and q in $\mathcal{C}$, denoted by $P_{\mathcal{C}}(p, q)$, as the shortest path from p to q that consists of arcs in $\{A_1, \ldots, A_n\} \cup \{B_1, \ldots, B_n\}$ and line segments in $\{\overline{a_1 b_1}, \ldots, \overline{a_{n-1} b_{n-1}}\}$. The *stretch factor* of a chain $\mathcal{C}$ is then given by the maximum value of $\frac{|P_{\mathcal{C}}(p,q)|}{|D_{\mathcal{C}}(p,q)|}$ over all terminals p and q of $\mathcal{C}$. Xia [100] establishes the following upper bound.

Theorem 3 (Theorem 1 [100]). *The stretch factor of a chain of disks $\mathcal{C}$ in the plane is less than $\rho = 1.998$.*

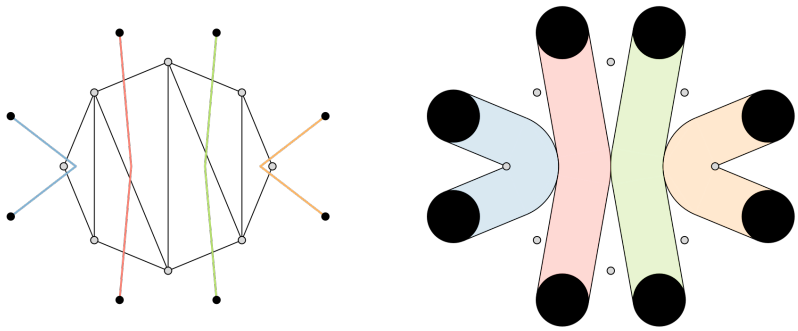
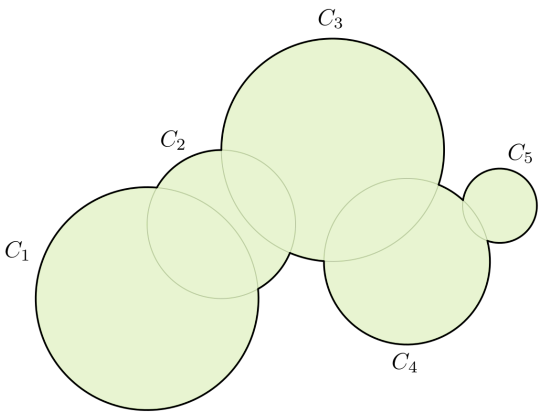
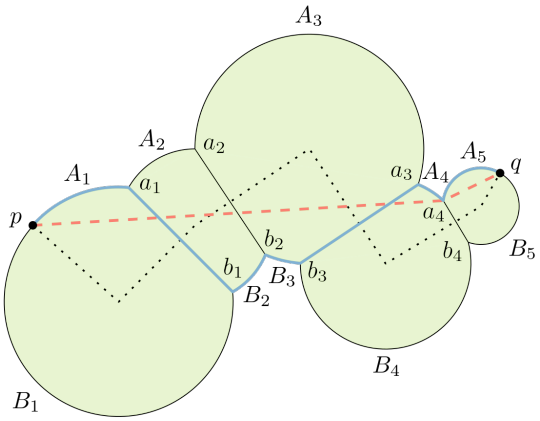


Figure 6.4: A construction for which the minimum separation factor approaches $\pi/2$.

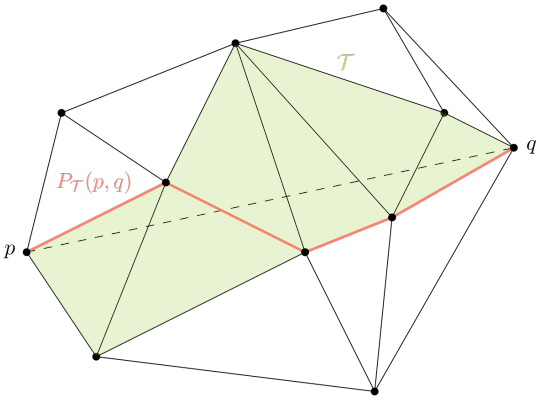


(a) A chain of disks $\mathcal{C} = (C_1, C_2, C_3, C_4, C_5)$.

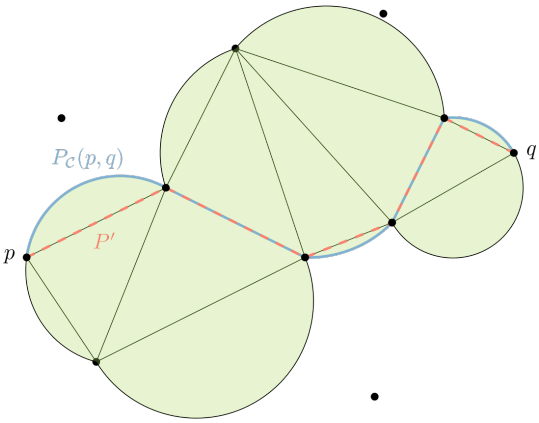


(b) For two terminals p and q of $\mathcal{C}$, the centered polyline between p and q is the dotted polyline, $D_{\mathcal{C}}(p, q)$ is the red dashed polyline and $P_{\mathcal{C}}(p, q)$ is the blue path.

Figure 6.5: Illustrations for the concepts and notation used by Xia [100].



(a) The sequence of triangles $\mathcal{T}$ through D intersected by $\overrightarrow{pq}$ and the shortest path $P_{\mathcal{T}}(p, q)$.



(b) The construction of path P' from $P_{\mathcal{C}}(p, q)$.

Figure 6.6: Illustrations for the restricted stretch factor of a Delaunay triangulation D.

Let D be a Delaunay triangulation on a set of points S and consider two arbitrary points $p, q \in S$. The ray $\overrightarrow{pq}$ intersects a sequence of triangles $\mathcal{T}$, whose union forms a simple polygon (see Figure 6.6(a)). Let $P_{\mathcal{T}}(p, q)$ denote the shortest path from p to q in $\mathcal{T}$, that is, the shortest path from p to q in D that is contained by the simple closed polygon induced by the triangles in $\mathcal{T}$. We then define the *restricted stretch factor* of a Delaunay triangulation as the maximum value of $\frac{|P_{\mathcal{T}}(p,q)|}{d(p,q)}$ over all points $p, q \in S$. We can now use the result by Xia to prove the following lemma.

Lemma 8. *The restricted stretch factor of a Delaunay triangulation D on a set of points S in the plane is less than $\rho = 1.998$.*

Proof. Consider two arbitrary points $p, q \in S$. Let $\mathcal{T}$ be the sequence of triangles in D crossed by the ray $\overrightarrow{pq}$ and let $\mathcal{C} = (C_1, C_2, \ldots, C_n)$ be the corresponding sequence of circumcircles of the triangles in $\mathcal{T}$. Clearly $\mathcal{C}$ is a chain and p, q are terminals of $\mathcal{C}$. By construction, the boundaries of every two consecutive disks in $\mathcal{C}$ intersect exactly in the common points of their corresponding triangles in $\mathcal{T}$. Therefore, a_i and b_i are points in S for all $1 \leq i \leq n$. Since each circumcircle C_i passes through the points a_i and b_i and by definition does not contain any other points of S in its

interior, it follows from the properties of a Delaunay triangulation that a_i and b_i are connected by an edge in D or represent the same point. The same argument holds for the points a_{i-1}, a_i and b_{i-1}, b_i for $1 \leq i \leq n$.

Consider the path $P_{\mathcal{C}}(p,q)$. We modify this path to obtain a path P' in $\mathcal{T}$ as follows (see Figure 6.6(b)). For each arc A_i in $P_{\mathcal{C}}(p,q)$, we add the edge between a_{i-1} and a_i to P', and for each arc B_i, we add the edge between b_{i-1} and b_i for $1 \leq i \leq n$. Each line segment $\overline{a_i b_i}$ in $P_{\mathcal{C}}(p,q)$ is simply represented by the edge between a_i and b_i in P'. Clearly P' is a path between p and q in $\mathcal{T}$ and has length at most $|P_{\mathcal{C}}(p,q)|$. Since $\overrightarrow{pq}$ traverses the circles in $\mathcal{C}$ in order, we have that $D_{\mathcal{C}}(p,q) = \overline{pq}$. Hence, by Theorem 3, the restricted stretch factor of a Delaunay triangulation is at most $\frac{|P'|}{d(p,q)} \leq \frac{|P_{\mathcal{C}}(p,q)|}{|D_{\mathcal{C}}(p,q)|} < \rho$. □

For the MDGE problem, we consider a Delaunay triangulation D on the vertices in V and the obstacles in O. We use the upper bound on the restricted stretch factor of a Delaunay triangulation to derive the same upper bound on the minimum separation factor of D.

Theorem 4. *The minimum-separation factor of a Delaunay triangulation D on the set of points $V \cup O$ is less than $\rho = 1.998$.*

Proof. To prove that ρ is an upper bound on the minimum separation factor of D, we need to prove that if $|uv| \geq \rho \cdot \tau_{sep}(u,v)$ for each edge (u,v) in D, we have that $d(p,q) \geq \tau_{sep}(p,q)$ for all $p, q \in V \cup O$. Consider two arbitrary points $p, q \in V \cup O$. Let $\mathcal{T}$ be the sequence of triangles intersected by the ray $\overrightarrow{pq}$ and let P' be the path between p and q in $\mathcal{T}$ using the same construction as in the proof of Lemma 8.

We argue that the total thickness of the shortest paths intersecting path P' is an upperbound for the total thickness of the paths separating p and q, that is,

$$\sum_{(u,v) \in P'} \tau_{sep}(u,v) \geq \tau_{sep}(p,q).$$

To prove this, it suffices to prove that if a link of the shortest path of an edge separates p and q, it must also intersect path P'. To this end, consider an edge $e \in E$ with shortest homotopic path π and let $l \in \pi$ be a link of π that separates p and q (see Figure 6.7). Suppose, towards a contradiction, that l does not intersect P'. Since l separates p and q, it must intersect line segment $\overline{pq}$. Furthermore, since π is the shortest homotopic path, l connects two distinct points $u, v \in V \cup O$, where at least one of u and v is enclosed by $\overline{pq}$ and P'. Assume, without loss of generality, that it is point u. Since both $\overline{pq}$ and P' are contained by $\mathcal{T}$, u must lie in the interior of $\mathcal{T}$, which contradicts the empty circumcircle property of the Delaunay triangulation. Hence,

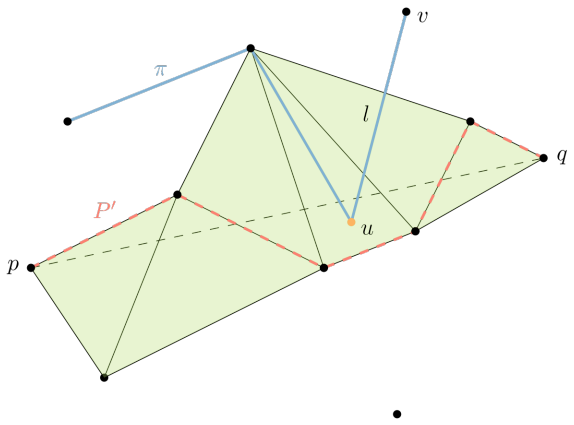


Figure 6.7: An illustration for the proof of Theorem 4.

link l must intersect path P' and, by Lemma 8, we have that

$$d(p,q) \geq \frac{|P'|}{\rho} = \sum_{(u,v)\in P'} \frac{|uv|}{\rho} \geq \sum_{(u,v)\in P'} \frac{\rho \cdot \tau_{sep}(u,v)}{\rho} \geq \tau_{sep}(p,q).$$

□

6.4 The linear program

By Theorem 4, it suffices to impose minimum-separation constraints on the edges of a Delaunay triangulation on the vertices and obstacles if we increase the minimum required separation of the constraints by a factor 1.998. Now we have a constrained optimization problem with a linear number of constraints, we still need a method to solve it. To this end, we observe that the optimization problem closely resembles the problem of removing overlap among a set of geometric shapes in the plane, such as squares or circles. In particular, both problems aim to minimize displacement while imposing constraints on the minimum required distance between pairs of objects.

Our approach for displacing the obstacles is based on the linear program by Meulemans [64], which solves the minimum-displacement overlap removal problem for a set of diamonds. They impose orthogonality constraints on the diamonds to maintain the original x- and y-orders of their centers. The constraint that two diamonds must be disjoint is specified using the Manhattan distance between their centers, as its unit circle is exactly a diamond. Although the Manhattan distance function involves absolute values, which are not linear, the preservation of the orthogonal order linearizes the computation. The objective function minimizes the total displacement of the diamonds, where displacement is measured using a convex polyhedral distance function δ. The unit 'circle' of δ, that is, the set of points at a distance of 1 from the origin, forms a convex polygon centered around the origin. Let C_δ contain, for each side of this polygon, the point on the segment that is closest to the origin. The polyhedral distance $\delta(p,q)$ between two points p and q can then be computed as $\max_{c\in C_\delta} c \cdot \frac{q-p}{\|c\|^2}$. Two well-known polyhedral distance functions are the Manhattan distance $L_1(p,q) = |x_p - x_q| + |y_p - y_q|$ and the Chebyshev distance $L_\infty(p,q) = \max(|x_p - x_q|, |y_p - y_q|)$, where $p = (x_p, y_p)$ and $q = (x_q, y_q)$. Although the minimum-displacement overlap removal problem is NP-hard for squares, rotating them by 45 degrees suddenly makes the problem tractable. This is because the orthogonality constraints induce a convex feasible space for placing a diamond with respect to another, which does not hold for squares (see Figure 6.8).

We adapt the linear program by Meulemans [64] to compute the new obstacle positions under the Delaunay-approximated minimum-separation constraints. Let $O = \{\omega_1, \ldots, \omega_{n_o}\}$ be the set of obstacles. For ease of notation, we use x_i and y_i instead of x_{ω_i} and y_{ω_i} to denote the x- and y-coordinate of obstacle ω_i for all $1 \leq i \leq n_o$. Consider a Delaunay triangulation D on the

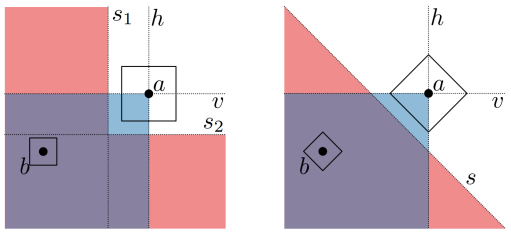


Figure 6.8: Feasible space (purple) for the placement of b with respect to a, which is nonconvex for squares (left) but convex for diamonds (right). Adapted from Meulemans [64].

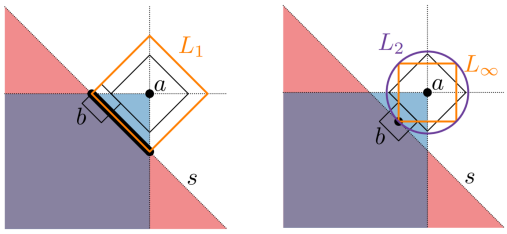


Figure 6.9: Optimal placement of b with respect to a, which can be anywhere on the thick black line for L_1 (left) but is unique and the same for L_∞ and L_2 (right). Adapted from Meulemans [64].

vertices in V and the obstacles in O. We use the polyhedral distance function $\delta = L_\infty$ to measure the displacement of the obstacles, as it is mentioned in [64] that this tends to give better results than the L_1 metric. This is because the unit circle of the Chebyshev distance is opposite to the unit circle of the Manhattan distance used for the constraints. As a result, the placement of two obstacles with respect to each other is unique and achieves minimum Euclidean distance (see Figure 6.9). The new positions of the obstacles are computed using the following linear program.

Variables

For each obstacle ω_i with $1 \leq i \leq n_o$, we introduce the variables x'_i, y'_i and d_i. The coordinates (x'_i, y'_i) represent the obstacle's new position and d_i its displacement $L_\infty((x_i, y_i), (x'_i, y'_i))$.

Objective function

The goal is to minimize the total obstacle displacement measured using the L_∞ metric. Hence, the objective function is

$$\text{minimize} \sum_{i=1}^{n_o} d_i.$$

Constraints

Meulemans adds constraints to preserve the orthogonal x- and y-orders on all objects. However, since the vertices in the MDGE problem are fixed, it would be too restrictive to maintain the orthogonal order on all vertices and obstacles. Furthermore, the primary purpose of imposing an orthogonality constraint on a pair of points is to make the feasible space for solving the corresponding minimum-separation constraint convex. Hence, since we only impose minimum-separation constraints on the edges in D, we propose to also limit the orthogonality constraints to the edges in D. Similar to the disjointness constraints for diamonds, the minimum-separation constraints are specified using the Manhattan distance. Since the Manhattan distance overestimates the Euclidean distance by a factor $\sqrt{2}$, we need to scale the minimum required separations by a factor of $\sqrt{2}$ to ensure that the constraints on the Euclidean distance are satisfied. This factor of $\sqrt{2}$ comes on top of the 1.998 approximation factor, thereby establishing a $1.998\sqrt{2}$-approximation of the minimum-separation constraints with preservation of the orthogonal order. For a pair of points connected by an edge in D, we can compute their new Manhattan distance using a linear constraint as we know their orthogonal order. We only present the constraints for pairs of points p and q such that p is to the left and below q, the other constraints are defined analogously. For each pair of obstacles ω_i, ω_j with $x_i \leq x_j$ and $y_i \leq y_j$ that are connected by an edge in D, we add the orthogonal order constraints

$$x'_i \leq x'_j \text{ and } y'_i \leq y'_j,$$

and the minimum-separation constraint

$$x'_j - x'_i + y'_j - y'_i \geq 1.998\sqrt{2} \cdot \tau_{sep}(\omega_i, \omega_j).$$

Similarly, for each obstacle ω_i and vertex $v \in V$ with $x_i \leq x_v$ and $y_i \leq y_v$ that are connected by an edge in D, we add the orthogonal order constraints

$$x'_i \leq x_v \text{ and } y'_i \leq y_v,$$

and the minimum-separation constraint

$$x'_i - x_v + y'_i - y_v \geq 1.998\sqrt{2} \cdot \tau_{sep}(\omega_i, v).$$

Finally, we must add constraints to compute the values of the displacement variables d_i such that $d_i = L_\infty((x_i, y_i), (x'_i, y'_i)) = \max(|x_i - x'_i|, |y_i - y'_i|)$ for all $1 \leq i \leq n_o$. Let $C_i = \{x_i - x'_i, x'_i - x_i, y_i - y'_i, y_i - y'_i\}$ be the set consisting of all possible values of d_i. By enforcing each d_i to be at

least c for each $c \in C_i$, it is at least the maximum. Since the objective function minimizes the sum over all d_i's, the value of d_i becomes equal to the maximum. Hence, for all $1 \leq i \leq n_o$ and $c \in C_i$, we add the constraint

$$d_i \geq c.$$

Correctness

Assume that the Delaunay triangulation D stayed the same after displacing the obstacles. Then, by Lemma 6 and Theorem 4, a solution that satisfies the constraints admits a planar thick embedding of the graph. As already mentioned, the placement of two points with respect to each other is unique and achieves minimum Euclidean distance. Therefore, the solution also minimizes the Euclidean distance and thus approximates the optimal obstacle positions with preserved orthogonal order by a factor $1.998\sqrt{2}$.

For each obstacle ω_i with $1 \leq i \leq n_o$, we introduce three variables and $|C_\delta|$ constraints to compute its displacement d_i. There are a total of $\mathcal{O}(|V| + n_o)$ orthogonal order and minimum-separation constraints, as we only add them for the edges in D. Hence, we can compute the new obstacle positions using a linear program with $\mathcal{O}(n_o)$ variables and $\mathcal{O}(|V| + n_o)$ constraints.

Theorem 5. *The linear program computes a $1.998\sqrt{2}$-approximation of the optimal obstacle positions with preserved orthogonal order under the assumption that the Delaunay triangulation on the vertices and obstacles stayed the same.*

6.5 The algorithm

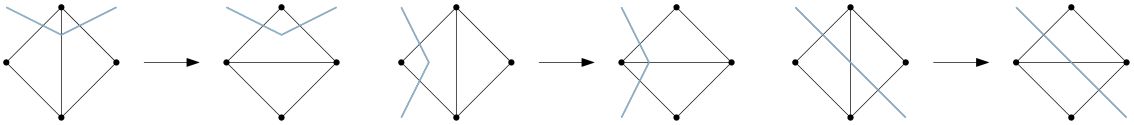
Our algorithm consists of two main components: displacing the obstacles to create enough space for the thick edges, and growing thick homotopic edges. The input edges are required to be the shortest homotopic paths, which otherwise can be computed efficiently using the method of [9] or [25]. Let D be a Delaunay triangulation on the vertices in V and the obstacles in O. We represent the homotopy of each edge $e_i \in E$ as the *crossing sequence* of its input path π_i through D, which is the sequence of Delaunay edges crossed by the path.

The minimum-separation constraints on the edges in D can be trivially computed from the crossing sequences. We compute the new obstacle positions using the linear program provided in Section 6.4. After displacing the obstacles, the Delaunay triangulation may have changed and we may need to add new minimum-separation constraints to the linear program. However, the crossing sequences may no longer be valid. Therefore, we need to dynamically update the Delaunay triangulation to maintain the crossing sequences of the edges. Recall that two paths are homotopic under displacement if one can be continuously deformed into the other without passing over any of the vertices or obstacles at any point in time (see Chapter 3). Our definition models the displacement of each obstacle as a linear interpolation from its original position to its new position. For this purpose, we propose to maintain D using a kinetic data structure [18, 103], which dynamically flips edges of non-Delaunay triangles to keep the Delaunay triangulation valid while moving the obstacles along a straight line from their original to their new positions.

Upon flipping an edge of the Delaunay triangulation, it is straightforward to update the crossing sequences of the edges. In particular, there are three main cases for updating the crossing sequence of a path upon flipping a Delaunay edge (see Figure 6.10). These cases reason about the way in which the path traverses the quadrilateral with the flipped Delaunay edge as diagonal. Let c_1 and c_2 denote the Delaunay edges via which the path enters and leaves the quadrilateral, respectively, and let d denote the flipped diagonal. We distinguish the following cases:

1. If the crossing sequence of a path has subsequence (c_1, d, c_2) and, after flipping d, crossings c_1 and c_2 are part of the same triangle, we remove d and the subsequence becomes (c_1, c_2).

2. If the crossing sequence of a path has subsequence (c_1, c_2) and, after flipping d, crossings c_1 and c_2 are part of different triangles, we insert d and the subsequence becomes (c_1, d, c_2).



(a) Case 1: the edge does no longer cross the diagonal after flipping.

(b) Case 2: the edge crosses the diagonal only after flipping.

(c) Case 3: the edge crosses the diagonal before and after flipping.

Figure 6.10: The three main cases for updating the crossing sequence of an edge upon flipping a Delaunay edge, based on how it traverses the corresponding quadrilateral.

3. If the crossing sequence of a path has subsequence (c_1, d, c_2) and, after flipping d, crossings c_1 and c_2 are part of different triangles, the subsequence remains (c_1, d, c_2).

After having displaced the obstacles towards their new positions, the minimum-separation constraints on the Delaunay edges can be determined from the updated crossing sequences. Subsequently, we again displace the obstacles using the linear program to resolve the new constraints. This process continues until we have found a solution satisfying all new minimum-separation constraints, or no solution exists. Since this process may go on indefinitely, we propose to retain the constraints from previous iterations in the linear program, which in the worst case leads to a quadratic number of constraints. The downside of this strategy is that it may enforce unnecessary constraints, possibly making the optimization problem overly restrictive.

If we found a solution, by Lemma 6 and Theorem 4, we can construct a planar thick embedding of the graph. Hence, we can now compute the thick homotopic edges using the growing algorithm by Duncan et al. [23] described in Section 6.1, completing the algorithm.

Chapter 7

Experimental evaluation

In this chapter, we present a proof of concept for the algorithm described in Chapter 6. Because our main contribution is the displacement of the obstacles, we simplify the implementation of the algorithm for growing thick edges by only considering graphs with a maximum degree of one. This removes the challenge of avoiding overlap between thick edges incident on the same vertex, while still allowing us to evaluate the quality of obstacle displacement. We tested our implementation on various manually created datasets. Since the MDGE problem appears to be unexplored in previous research, we were unable to perform a comparative analysis against other algorithms.

7.1 Implementation

We implemented the algorithm using Python 3.12. The code of our implementation is available in our public Github repository[1]. All code is original and developed independently, except for the use of standard Python libraries. We briefly describe our implementations for the three main components of the algorithm: displacing the obstacles, computing shortest homotopic edges, and growing thick homotopic edges.

Displacing the obstacles

To compute the new obstacle positions, we solve the linear program described in Section 6.4 using Gurobi Optimization[2]. The presented set of constraints allows the obstacles to be placed on top of each other, but this is generally not desired and may lead to unexpected behavior. Therefore, we slightly adapt the orthogonality constraints to ensure disjointness as follows. If two obstacles connected by a Delaunay edge have the same x-coordinate, their new x-coordinates must also be equal and the orthogonality constraint remains the same. However, if their x-coordinates are different, we enforce a minimum x-difference of one. In particular, for two obstacles ω_i and ω_j with $x_i < x_j$, the new orthogonality constraint becomes $x'_i \leq x'_j - 1$. The orthogonality constraints on the y-coordinates are adapted in the same way.

To update the Delaunay triangulation upon displacing the obstacles, we do not use a kinetic data structure. Instead, to simplify the implementation, we simply place the obstacles at their new positions and then attempt to restore the Delaunay triangulation by flipping edges until all triangles satisfy the Delaunay property. This strategy is guaranteed to work when we still have a valid (not necessarily Delaunay) triangulation after displacing the obstacles. When obstacles are moved 'over' Delaunay edges, the triangulation may become invalid and our method may not work as expected. However, due to the orthogonality constraints, this rarely happens and we need only very few edge flips in practice. To prevent the algorithm from entering an infinite

[1] https://github.com/sashagielis/MDGE
[2] https://www.gurobi.com/

loop of displacing obstacles and recomputing constraints, we retain all minimum-separation and orthogonality constraints across successive iterations of the linear program.

Computing shortest homotopic paths

The algorithm for growing thick edges requires the shortest homotopic paths as input. Therefore, after having displaced the obstacles, we compute the shortest homotopic paths of the edges based on their (updated) crossing sequences using the algorithm by Lee and Preparata [55]. Given two points within a simple polygon, they determine the (unique) crossing sequence of the path through the triangulated polygon and compute the shortest path between the two points by growing a funnel through the corresponding sequence of triangles. Starting from the source point, the crossings of the crossing sequence are handled in order and the funnel is maintained to represent the union of shortest paths from the source point up to that crossing. After having processed all crossings, the shortest path between the two points can be obtained from the funnel.

The paper describes the algorithm for two points within a simple polygon, in which case the homotopy of the path is fixed, while we need to compute the shortest homotopic path between two points in the plane. However, the algorithm of [55] works for any triangulation with all triangle vertices on its boundary, even when the path traverses the same triangle multiple times. Therefore, since the sequence of triangles induced by the crossing sequence of an edge satisfies this property, we can directly use the algorithm to compute the shortest homotopic paths.

Growing thick homotopic edges

We implemented the growing algorithm by Duncan et al. [23] explained in Section 6.1 to compute the thick homotopic edges. However, as we only consider graphs with a maximum degree of one for this proof of concept, we did not implement the extension towards arbitrary graphs. During the growth process, we maintain for each elbow bundle the next split and merge event it will cause, independent of other bundles. While the paper suggests to also maintain the next events for all straight bundles, we found this unnecessary for our implementation. As the edge thicknesses are predefined, we only consider event times between 0 and 1, denoting the fraction of the thickness of the edges. We do not compute the exact times of the next split and merge events but approximate them using binary search, which works well in practice. Since our method for displacing the obstacles ensures that there is enough space to draw the thick edges, we do not need to compute stop events. Instead, when the event queue is empty, the edges have reached their required thickness and we terminate the algorithm.

The paper provides pseudocode for the various operations for maintaining the bundles in the compact routing structure. However, we noticed a mistake in the code for the *divide* operation of the algorithm. We refer to Appendix A for the corrected pseudocode.

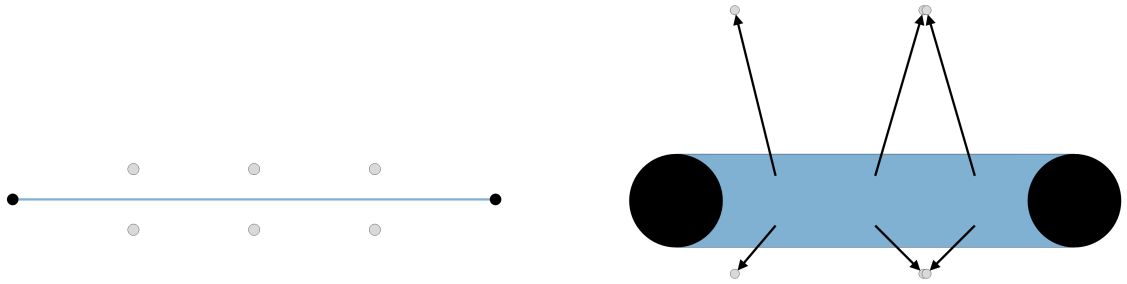
7.2 Datasets

We tested our algorithm on various datasets, all of which were manually created using IPE[3] and can be directly read as input by our implementation. Small instances were used to investigate what types of solution our algorithm generates. These simple instances allow us to reason about the optimal solution and analyze how our displacement strategy differs from the optimal approach. Additionally, we designed several larger datasets to more closely resemble real-world scenarios, allowing us to evaluate the practical relevance of the algorithm.

7.3 Results

We first study the type of solution that our algorithm produces. To this end, consider the instance in Figure 7.1(a), consisting of a single edge with three obstacles on either side. The most natural

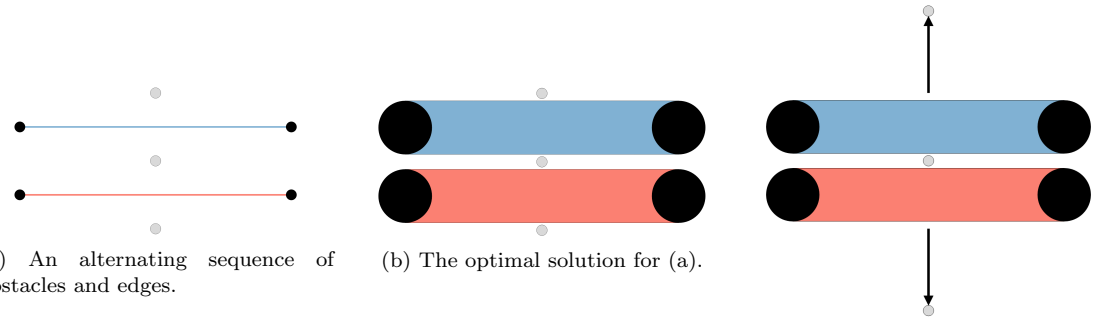
[3] https://ipe.otfried.org/



(a) Three pairs of obstacles separated by an edge.

(b) Our solution in which the obstacles are displaced diagonally instead of vertically.

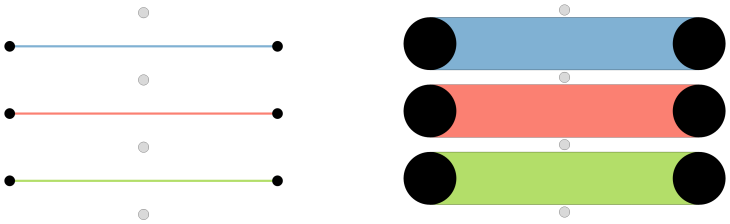
Figure 7.1: The type of displacement applied by the linear program.



(a) An alternating sequence of obstacles and edges.

(b) The optimal solution for (a).

(c) Our solution for (a).



(d) The instance of (a) extended by an extra edge and obstacle.

(e) The optimal solution for (d).

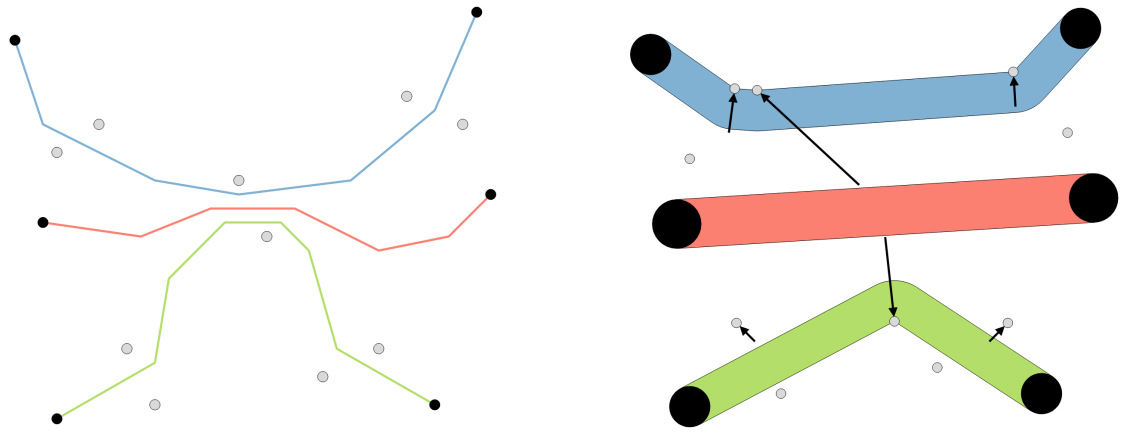
Figure 7.2: The orthogonality constraints highly restrict the solution space. While our algorithm can compute a solution for the instance in (a), the instance in (d) cannot be solved since the movements of the two center obstacles are restricted by their surrounding vertices.

solution would be to separate the obstacles by sliding them apart in a vertical direction, as this is also done in an optimal solution. However, as can be seen in Figure 7.1(b), our algorithm moves the obstacles diagonally. This can be explained as follows. The linear program uses the Manhattan distance to enforce the minimum-separation constraints. Due to the orthogonality constraints, each pair of obstacles with the same x-coordinate also have the same x-coordinate in the solution, and thus their Manhattan distance is exactly equal to the difference in y-coordinates. The objective function of the linear program uses the Chebyshev distance to minimize the displacement of the obstacles, which takes the maximum of the x- and y-differences. While the vertical displacement ensures that there is enough space between the two obstacles, the horizontal displacement does not affect the optimality of the solution as long as it is smaller. As a result, any displacement with a smaller x-difference than y-difference that satisfies the orthogonality constraints is an optimal solution to the linear program.

Although the orthogonality constraints of the linear program allow for an efficient computation of the new obstacle positions, they highly restrict the solution space. This is illustrated in Figure 7.2. The two considered instances involve an alternating sequence of obstacles and edges. While both can be solved without displacing any of the obstacles, the Delaunay-approximated minimum-separation constraints are not satisfied. For the instance in (a), the orthogonality constraints enforce the center obstacle to remain within the 'bounding box' of its four surrounding vertices. As a result, the algorithm displaces the other two obstacles vertically up and down to create enough separation. Consider extending the instance by another edge and obstacle as shown in (d). The two obstacles on either end of the alternating sequence can again be 'freely' displaced vertically up and down. However, now there are two center obstacles whose movement is restricted by their surrounding vertices. To resolve the minimum-separation constraint on these two obstacles, they need to be moved outside of their bounding boxes or change their x-order. Hence, it becomes impossible to separate the two obstacles while satisfying the orthogonality constraints, and thus our algorithm fails to find a solution in this case.

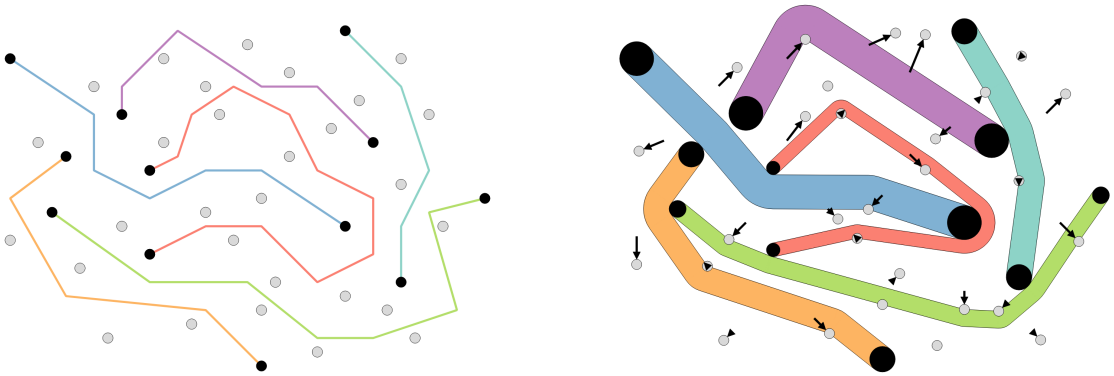
To assess the practical applicability of our algorithm, we demonstrate its performance on various datasets. We discuss a selection of the solutions generated by our algorithm and refer to Appendix B for the results on the remaining datasets (except for instance 6). Consider the instance in Figure 7.3(a), consisting of three edges and five pairs of obstacles. The linear program adds a minimum-separation constraint on each of these pairs. Clearly, the constraint on the two obstacles in the center is the most severe, as these obstacles are separated by all three edges and there is only little space between them. Consequently, our algorithm applied the largest amounts of displacement to the two center obstacles, as shown in Figure 7.3(b).

See Figure 7.4 for a larger example in which the displacement is more evenly distributed across the obstacles. Although the obstacles are displaced in various directions, the orthogonality constraints ensure that their relative positions with respect to each other are maintained. As can be seen, the obstacles leave more space than necessary for drawing the thick edges. However, while it is clearly possible to draw the graph with thicker edges, the algorithm does not allow this. In fact, even a slight increase in edge thickness would already make both instances 1 and 2 infeasible. Since, as demonstrated in Figure 7.2, this limitation arises from the vertices being fixed, allowing the vertices to be displaced would make the orthogonality constraints less restrictive, potentially enabling the specification of thicker edges. To test this hypothesis, we modified the linear program to permit vertex displacement and then solved instance 2 for progressively larger edge thicknesses proportional to the original thicknesses of the edges. As shown in Figure 7.6, the new implementation is capable of solving the instance with up to twice the original edge



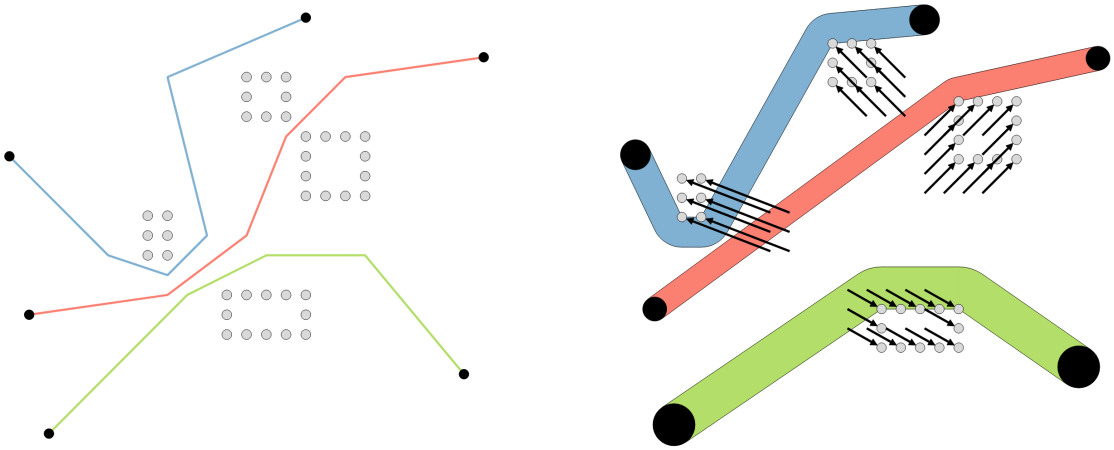
(a) Input. (b) Output.

Figure 7.3: Result for instance 1.



(a) Input. (b) Output.

Figure 7.4: Result for instance 2.



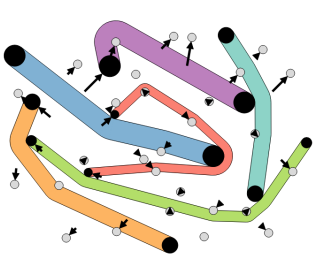
(a) Input. (b) Output.

Figure 7.5: Result for polygonal instance 1.

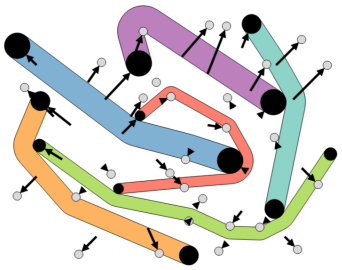
thickness. The applied displacements separate the points by gradually 'pushing' them outwards. Hence, allowing the vertices to be displaced clearly relaxes the orthogonality constraints, thereby significantly expanding the solution space of the linear program.

To illustrate the extent to which we overestimate the optimal minimum-separation constraints, we show by how much we can still increase the thicknesses of the edges after displacement. Starting from the solution depicted in Figure 7.6(f), we simply let the growing algorithm continue after reaching the required thickness. The resulting growth process of the edges is visualized in Figure B.5 of Appendix B. As can be seen, the edges managed to achieve almost twice their original thickness. This indicates that the point separation produced by our algorithm is almost a factor of two larger than necessary, which aligns with the approximation factor of 1.998 for the minimum-separation constraints on the Delaunay edges.

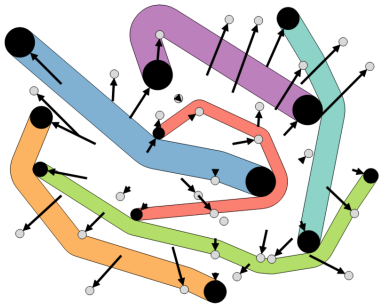
While our algorithm only works with point obstacles, practical applications often involve polygonal obstacles. We can still use our algorithm to handle the displacement of polygonal obstacles by representing each polygon as a set of point obstacles placed along its boundary. The density of the points determines how well the shape of the polygon is approximated. Since the orthogonality constraints of the linear program ensure that the x- and y-orders of the obstacles are maintained, our algorithm works best for rectilinear obstacles. To ensure that the polygons maintain their



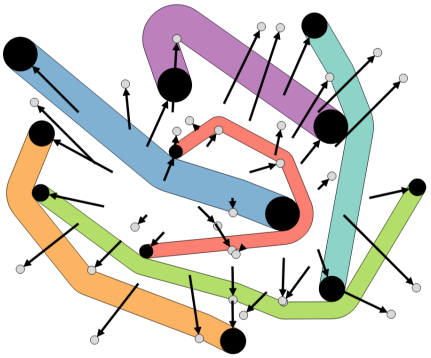
(a) $\tau(\cdot)$.



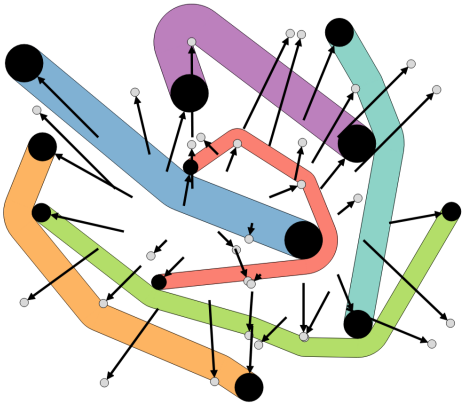
(b) $1.2\tau(\cdot)$.



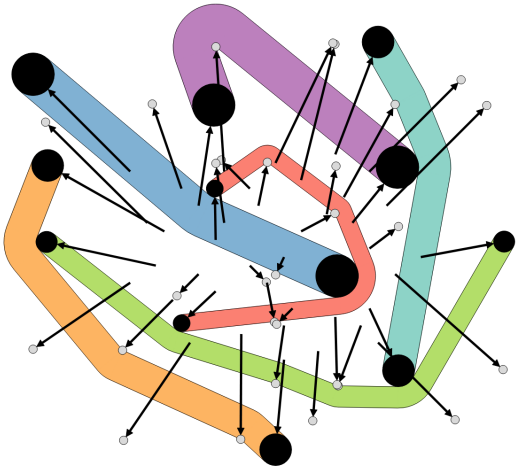
(c) $1.4\tau(\cdot)$.



(d) $1.6\tau(\cdot)$.



(e) $1.8\tau(\cdot)$.



(f) $2\tau(\cdot)$.

Figure 7.6: Results for instance 2 with progressively larger edge thickness and displaced vertices, where $\tau(\cdot)$ denotes the original thickness of the edges.

Instance	$\|O\|$	Number of iterations	Min-sep constraints	Running time (seconds)		
				Displace	Grow	Total
1	10	1	32	0.008	0.082	0.095
2	29	2	99, 105	0.049	1.035	1.097
3	15	1	57	0.015	0.505	0.529
4	22	2	64, 75	0.049	0.275	0.333
5	30	2	77, 90	0.038	0.471	0.518
6	1000	2	2001, 2079	5.207	0.445	5.818
Poly 1	38	1	114	0.042	0.351	0.407
Poly 2	22	1	68	0.047	0.300	0.366

Table 7.1: Information of the results for our datasets.

original shape, we add constraints to the linear program to ensure that each pair of consecutive points along the boundary of a polygon preserve their exact relative positions. Figure 7.5 shows an example where each point belonging to the same polygonal obstacle is applied a similar displacement such that the represented polygons maintain their original shape.

The results from applying our algorithm on the datasets are presented in Table 7.1. For each instance, the linear program needs at most two iterations to separate the obstacles. We can also see that the second iteration introduces very few new minimum-separation constraints, indicating that the Delaunay triangulation needed very few updates thanks to the orthogonality constraints. To measure the running times, we took the average over 100 runs of our algorithm. As can be seen, our algorithm is able to solve all instances efficiently. We also constructed a dataset with 1000 obstacles, which is essentially an extended version of the instance shown in Figure 7.1(a) with 500 obstacles on either side of the edge. The algorithm required two displacement iterations with respectively 2001 and 2079 minimum-separation constraints, but still managed to solve this large instance in roughly 6 seconds. Overall, we can conclude that our algorithm is both efficient and capable of producing practical solutions for the tested datasets.

Chapter 8

Discussion

In this thesis, we have investigated the MDGE problem from both a theoretical and practical perspective. We formulated two research questions, which we answer in this chapter. Additionally, we provide open problems and possible directions for future work.

To study the computational complexity of the MDGE problem, we focused on minimizing the maximum obstacle displacement, as this typically leads to a simpler optimization problem than minimizing the total displacement. We first examined the one-dimensional MDGE problem, for which we provided an efficient algorithm in Chapter 4. In two dimensions, however, the problem becomes significantly more challenging. In Chapter 5, we showed that, when allowing the thick edges in the output to overlap each other, the MDGE problem is NP-hard. Although the complexity result is not proven for the original MDGE problem, the fact that a slight modification of the problem already makes it NP-hard suggests that the original problem is likely NP-hard as well. Hence, while our result does not directly solve the posed research question, it does illustrate the inherent complexity of the MDGE problem.

In Chapter 6, we proposed an algorithm for the MDGE problem, where the goal is to minimize the total obstacle displacement. The complexity of the MDGE problem became apparent during the process of algorithm design, as it was not directly evident how to check the validity of a solution. In particular, merely verifying whether we can construct a planar thick embedding of a graph under a given set of static obstacles is not trivial. The growing algorithm by Duncan et al. [23] can be used to determine whether a solution is valid but does not provide a practical method for checking feasibility. Instead, we observed that the MDGE problem can be formulated as a constrained optimization problem, with constraints specifying the minimum required separation between pairs of points. While this provides a simple method to verify whether a solution exists, the constructed problem includes a constraint on each pair of points, which does not allow for a practical algorithm. We were able to reduce the number of minimum-separation constraints from quadratic to linear by only including constraints on the edges of a Delaunay triangulation on the vertices and obstacles. This reduction in constraints comes at the cost of increasing the minimum required separation of each imposed constraint by a factor of 1.998.

Since we represent the homotopy of an edge using its crossing sequence through the Delaunay triangulation, the Delaunay-approximated minimum-separation constraints can be trivially computed from the crossing sequences of the edges. To actually displace the obstacles, we use a result on minimum-displacement overlap removal by Meulemans [64]. Their linear program is adapted to compute the new obstacle positions under the Delaunay-approximated minimum-separation constraints. The constraints on the orthogonal order of the points induce a convex feasible space, which allows the linear program to be solved efficiently. Due to the use of the Manhattan distance for specifying the minimum-separation constraints, the minimum required separation of each constraint must be increased by an additional factor of $\sqrt{2}$, on top of the 1.998 approximation factor. After displacing the obstacles, the Delaunay triangulation may have changed and new minimum-separation constraints may need to be added to the linear program. Using a kinetic data structure, we can maintain the Delaunay triangulation throughout the displacement of the

obstacles from their original to their new positions. We showed that, upon flipping an edge of the Delaunay triangulation, it is trivial to update the crossing sequences of the edges and thus compute the new minimum-separation constraints. The process of displacing obstacles and computing new constraints may go on indefinitely. As a result, the algorithm is not guaranteed to terminate. By keeping the constraints from previous iterations of the linear program, we increase our chances of finding a solution.

We implemented our algorithm and tested its performance in Chapter 7. From the conducted experiments, it becomes clear that, due to the Delaunay-approximated minimum-separation constraints, the linear program attempts to create significantly more space than necessary for drawing the thick edges. However, since the orthogonality constraints highly restrict the solution space, this sometimes prevents the algorithm from finding a solution while one does exist. Additionally, since the minimum-separation constraints are specified using the Manhattan distance and the objective function measures the displacement of the obstacles using the Chebyshev distance, the algorithm typically follows a different displacement strategy than an optimal solution and the generated results may appear counter-intuitive. Despite the potentially infinite process of obstacle displacement, in practice, the algorithm always terminates and typically needs only one or two displacement iterations. Furthermore, all our datasets could be solved efficiently. Overall, although the orthogonality constraints limit the solutions explored by the algorithm, we can conclude that our algorithm is a practical and efficient method to solve the MDGE problem.

Open problems and future work

We studied the computational complexity of the MDGE problem by only considering the objective of minimizing the maximum obstacle displacement. However, the proposed algorithm for the MDGE problem minimizes the total displacement of all obstacles. Therefore, it would be useful to examine whether similar theoretical results can be established when the goal is to minimize the total displacement. Additionally, we only proved NP-hardness for the variant of the MDGE problem that allows overlap among the thick edges in the solution. It remains an open problem whether the MDGE problem is also NP-hard without this modification.

The approximation factor of 1.998 on the Delaunay-approximated minimum-separation constraints directly follows from the upper bound on the restricted stretch factor of a Delaunay triangulation. Therefore, the algorithm can be improved by reducing this bound. Although the orthogonality constraints of the linear program allow for an efficient computation of the new obstacle positions, they also restrict the solutions explored by our algorithm. We showed that the orthogonality constraints can be relaxed by allowing the vertices to be displaced, thereby expanding the solution space. While the variant of the problem with displaced vertices may prove useful in some applications, it differs from our original motivation, being the design of flow maps depicting flows between fixed locations on a map. A direction for future work would be to design an algorithm for the MDGE problem whose efficient computation does not rely on constraints on the orthogonal order of the points, or at least on constraints that are less restrictive.

Our work can be extended in various ways. First of all, the implementation can be extended to work for arbitrary graphs. This involves drawing each vertex as the smallest disk such that its incident thick edges do not overlap outside of that disk. Another extension involves making our algorithm more applicable to real-world scenarios. Flow maps typically include polygonal obstacles such as countries, building or lakes. Therefore, the restriction to point obstacles does not allow for practical applications in flow map design. We showed that we can use our algorithm to displace a set of polygonal obstacles by representing each polygon as a set of point obstacles along their boundary and adding extra constraints to the linear program. However, this approach was not thoroughly tested. Furthermore, displacement may not always be enough as sole method for distortion as some maps may require the obstacles to be deformed. Additionally, the growing algorithm for computing thick edges needs to be adapted to work with polygonal obstacles.

Bibliography

[1] T. Ai, X. Zhang, Q. Zhou, and M. Yang. A vector field model to handle the displacement of multiple conflicts in building generalization. *International Journal of Geographical Information Science*, 29(8):1310–1331, 2015. 13

[2] J. Alam, S.G. Kobourov, and S. Veeramoni. Quantitative Measures for Cartogram Generation Techniques. *Computer Graphics Forum*, 34(3):351–360, 2015. 11, 12

[3] G. Barequet, M.T. Goodrich, and C. Riley. Drawing Planar Graphs with Large Vertices and Thick Edges. *Journal of Graph Algorithms and Applications*, 8(1):3–20, 2004. 5, 6

[4] T. Barkowsky, L.J. Latecki, and K. Richter. Schematizing Maps: Simplification of Geographic Shape by Discrete Curve Evolution. In *Spatial Cognition II: Integrating Abstract Theories, Empirical Studies, Formal Methods, and Practical Applications*, pages 41–53. Springer Berlin Heidelberg, 2000. 12

[5] M. Basaraner. A zone-based iterative building displacement method through the collective use of Voronoi Tessellation, spatial analysis and multicriteria decision making. *Boletim de Ciências Geodésicas*, 17(2):161–187, 2011. 13

[6] G. Di Battista, P. Eades, R. Tamassia, and I.G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Computational Geometry*, 4(5):235–282, 1994. 5

[7] J. van den Berg, M. Stilman, J. Kuffner, M. Lin, and D. Manocha. *Path Planning among Movable Obstacles: A Probabilistically Complete Approach*, pages 599–614. Springer Berlin Heidelberg, 2010. 7

[8] M. de Berg and A. Khosravi. Optimal Binary Space Partitions for Segments in the Plane. *International Journal of Computational Geometry & Applications*, 22(03):187–205, 2012. 23

[9] S. Bespamyatnikh. Computing homotopic shortest paths in the plane. *Journal of Algorithms*, 49(2):284–303, 2003. 38

[10] K. Buchin, B. Speckmann, and K. Verbeek. Flow Map Layout via Spiral Trees. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2536–2544, 2011. 14, 15

[11] K. Buchin, B. Speckmann, and S. Verdonschot. Evolution Strategies for Optimizing Rectangular Cartograms. In *Geographic Information Science*, GIScience 2012, pages 29–42. Springer Berlin Heidelberg, 2012. 11

[12] F. Chen, L. Piccinini, P. Poncelet, and A. Sallaberry. Node Overlap Removal Algorithms: an Extended Comparative Study. *Journal of Graph Algorithms and Applications*, 24(4):683–706, 2020. 8

[13] P.C. Chen and Y.K. Hwang. Practical Path Planning among Movable Obstacles. In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, pages 444–449. IEEE Computer Society, 1991. 7

[14] Z. Chen, Y. Shen, X. Lv, Q. Qin, and X. Chen. Variable-Scale Visualization of High-Density Polygonal Buildings on a Tile Map. *ISPRS International Journal of Geo-Information*, 11(10):505, 2022. 9

[15] L.P. Chew. Planning the Shortest Path for a Disc in $O(n^2 \log n)$ Time. In *Proceedings of the First Annual Symposium on Computational Geometry*, SCG '85, pages 214–220. Association for Computing Machinery, 1985. 8

[16] L.P. Chew. There are Planar Graphs Almost as Good as the Complete Graph. *Journal of Computer and System Sciences*, 39(2):205–219, 1989. 33

[17] S. Cicerone and M. Cermignani. Fast and Simple Approach for Polygon Schematization. In *Computational Science and Its Applications – ICCSA 2012*, ICCSA 2012, pages 267–279. Springer Berlin Heidelberg, 2012. 12

[18] O. Devillers and P.M.M. de Castro. State of the Art: Updating Delaunay Triangulations for Moving Points. Research Report RR-6665, INRIA, 2008. 38

[19] T. van Dijk, A. van Goethem, J. Haunert, W. Meulemans, and B. Speckmann. Accentuating Focus Maps via Partial Schematization. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '13, pages 428–431. Association for Computing Machinery, 2013. 12, 14, 15

[20] T.C. van Dijk, A. van Goethem, J. Haunert, W. Meulemans, and B. Speckmann. Map Schematization with Circular Arcs. In *Geographic Information Science*, GIScience 2014, pages 1–17. Springer International Publishing, 2014. 12

[21] T.C. van Dijk and J. Haunert. Interactive focus maps using least-squares optimization. *International Journal of Geographical Information Science*, 28(10):2052–2075, 2014. 10

[22] D. Dorling. Area Cartograms: Their Use and Creation. In *The Map Reader: Theories of Mapping Practice and Cartographic Representation*, pages 252–260. John Wiley & Sons, Ltd, 2011. 11

[23] C. Duncan, A. Efrat, S. Kobourov, and C. Wenk. Drawing with Fat Edges. *International Journal of Foundations of Computer Science*, 17(5):1143–1163, 2006. 5, 6, 29, 30, 31, 39, 42, 49

[24] T. Dwyer, K. Marriott, and P.J. Stuckey. Fast Node Overlap Removal. In *Graph Drawing*, GD 2005, pages 153–164. Springer Berlin Heidelberg, 2006. 8

[25] A. Efrat, S.G. Kobourov, and A. Lubiw. Computing homotopic shortest paths efficiently. *Computational Geometry*, 35(3):162–172, 2006. 38

[26] D. Fairbairn and G. Taylor. Developing a variable-scale map projection for urban areas. *Computers & Geosciences*, 21(9):1053–1064, 1995. 9

[27] J. Fiala, J. Kratochvíl, and A. Proskurowski. Systems of distant representatives. *Discrete Applied Mathematics*, 145(2):306–316, 2005. 8

[28] A.M. Francis and J.B. Schneider. Using computer graphics to map origin-destination data describing health care delivery systems. *Social Science & Medicine*, 18(5):405–420, 1984. 1, 14

[29] G.W. Furnas. Generalized Fisheye Views. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '86, pages 16–23. Association for Computing Machinery, 1986. 9

[30] E.R. Gansner and Y. Hu. Efficient Node Overlap Removal Using a Proximity Stress Model. In *Graph Drawing: 16th International Symposium, GD 2008; Heraklion, Crete, Greece, September 2008; Revised Papers*, volume 5417 of *GD 2008*, pages 206–217. Springer Berlin Heidelberg, 2009. 8

[31] S. Gao, M. Jerrum, M. Kaufman, K. Mehlhorn, and W. Rülling. On Continuous Homotopic One Layer Routing. In *Proceedings of the Fourth Annual Symposium on Computational Geometry*, SCG '88, pages 392–402. Association for Computing Machinery, 1988. 17

[32] M. van Garderen, B. Pampel, A. Nocaj, and U. Brandes. Minimum-Displacement Overlap Removal for Geo-referenced Data Visualization. *Computer Graphics Forum*, 36(3):423–433, 2017. 8

[33] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979. 23

[34] M.T. Gastner and M.E.J. Newman. Diffusion-based method for producing density-equalizing maps. *Proceedings of the National Academy of Sciences*, 101(20):7499–7504, 2004. 11

[35] H. Gibson, J. Faith, and P. Vickers. A survey of two-dimensional graph layout techniques for information visualisation. *Information Visualization*, 12(3-4):324–357, 2013. 5

[36] D. Guo and X. Zhu. Origin-Destination Flow Data Smoothing and Mapping. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2043–2052, 2014. 14

[37] L. Harrie, L.T. Sarjakoski, and L. Lehto. A mapping function for variable-scale maps in small-display cartography. *Journal of Geospatial Engineering*, 4(2):111–123, 2002. 9

[38] L.E. Harrie. The Constraint Method for Solving Spatial Conflicts in Cartographic Generalization. *Cartography and Geographic Information Science*, 26(1):55–69, 1999. 13

[39] J. Haunert and L. Sering. Drawing Road Networks with Focus Regions. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2555–2562, 2011. 10, 14

[40] K. Hauser. Minimum Constraint Displacement Motion Planning. In *Proceedings of Robotics: Science and Systems*, 2013. 7

[41] K. Hayashi, M. Inoue, T. Masuzawa, and H. Fujiwara. A Layout Adjustment Problem for Disjoint Rectangles Preserving Orthogonal Order. *Systems and Computers in Japan*, 33(2):31–42, 2002. 8

[42] W. He and K. Marriott. Constrained Graph Layout. *Constraints An International Journal*, 3(4):289–314, 1998. 8

[43] J. Hopcroft and R. Tarjan. Efficient Planarity Testing. *Association for Computing Machinery*, 21(4):549–568, 1974. 5

[44] T.C. Hu, A.B. Kahng, and G. Robins. Optimal robust path planning in general environments. *IEEE Transactions on Robotics and Automation*, 9(6):775–784, 1993. 8

[45] H. Huang, Q. Guo, Y. Sun, and Y. Liu. Reducing Building Conflicts in Map Generalization with an Improved PSO Algorithm. *ISPRS International Journal of Geo-Information*, 6(5):127, 2017. 13

[46] X. Huang and W. Lai. Force-Transfer: A New Approach to Removing Overlapping Nodes in Graph Layout. In *Proceedings of the Twenty-Sixth Australasian Computer Science Conference*, volume 16 of *CRPIT*, pages 349–358. Australian Computer Society, 2003. 8

[47] P. Højholt. Solving Space Conflicts in Map Generalization: Using a Finite Element Method. *Cartography and Geographic Information Science*, 27(1):65–74, 2000. 13

[48] B. Jenny, D.M. Stephen, I. Muehlenhaus, B.E. Marston, R. Sharma, E. Zhang, and H. Jenny. Force-directed layout of origin-destination flow maps. *International Journal of Geographical Information Science*, 31(8):1521–1540, 2017. 14

[49] B. Jenny, D.M. Stephen, I. Muehlenhaus, B.E. Marston, R. Sharma, E. Zhang, and H. Jenny. Design principles for origin-destination flow maps. *Cartography and Geographic Information Science*, 45(1):62–75, 2018. 14

[50] M. Kaufmann and D. Wagner, editors. *Drawing Graphs: Methods and Models*. Springer Berlin Heidelberg, 2001. 5

[51] D. E. Knuth and A. Raghunathan. The Problem of Compatible Representatives. *SIAM Journal on Discrete Mathematics*, 5(3):422–427, 1992. 23

[52] I. Kostitsyna and V. Polishchuk. Simple Wriggling is Hard Unless You Are a Fat Hippo. *Theory of Computing Systems*, 50(1):93–110, 2012. 7, 8

[53] C. Koylu, G. Tian, and M. Windsor. Flowmapper.org: a web-based framework for designing origin–destination flow maps. *Journal of Maps*, 19(1):1996479, 2023. 14

[54] M. van Kreveld. Bold graph drawings. *Computational Geometry*, 44(9):499–506, 2011. 5, 6

[55] D. T. Lee and F. P. Preparata. Euclidean shortest paths in the presence of rectilinear barriers. *Networks*, 14(3):393–410, 1984. 42

[56] W. Li, T. Ai, Y. Shen, W. Yang, and W. Wang. A Novel Method for Building Displacement Based on Multipopulation Genetic Algorithm. *Applied Sciences*, 10(23):8441, 2020. 13

[57] Z. Li. Digital Map Generalization at the Age of Enlightenment: a Review of the First Forty Years. *The Cartographic Journal*, 44(1):80–93, 2007. 13

[58] Z. Li and P. Ti. Adaptive generation of variable-scale network maps for small displays based on line density distribution. *GeoInformatica*, 19(2):277–295, 2015. 10

[59] D. Lichtenstein. Planar Formulae and Their Uses. *SIAM Journal on Computing*, 11(2):329–343, 1982. 23

[60] Y. Liu, M. Zhang, S. Li, S. Yang, and F. Dong. A spatial contextual immune genetic algorithm to building cartographic displacement. *Transactions in GIS*, 27(4):1228–1262, 2023. 13

[61] M. Lonergan and C.B. Jones. An Iterative Displacement Method for Conflict Resolution in Map Generalization. *Algorithmica*, 30(2):287–301, 2001. 13

[62] W.A. Mackaness and R.S. Purves. Automated Displacement for Large Numbers of Discrete Map Objects. *Algorithmica*, 30(2):302–311, 2001. 13

[63] K. Marriott, P.J. Stuckey, V. Tam, and W. He. Removing Node Overlapping in Graph Layout Using Constrained Optimization. *Constraints An International Journal*, 8:143–171, 2003. 8

[64] W. Meulemans. Efficient Optimal Overlap Removal: Algorithms and Experiments. *Computer Graphics Forum*, 38(3):713–723, 2019. 8, 36, 37, 49

[65] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout Adjustment and the Mental Map. *Journal of Visual Languages & Computing*, 6(2):183–210, 1995. 8

[66] D. Nieuwenhuisen, A.F. van der Stappen, and M.H. Overmars. *An Effective Framework for Path Planning Amidst Movable Obstacles*, pages 87–102. Springer Berlin Heidelberg, 2008. 7

[67] T. Nishizeki and S. Rahman. *Planar Graph Drawing*. World Scientific, 2004. 5

[68] S. Nusrat, J. Alam, and S. Kobourov. Evaluating Cartogram Effectiveness. *IEEE Transactions on Visualization and Computer Graphics*, 24(2):1077–1090, 2018. 11

[69] S. Nusrat and S. Kobourov. The State of the Art in Cartograms. *Computer Graphics Forum*, 35(3):619–642, 2016. 11

[70] J.M. Olson. Noncontiguous area cartograms. *The Professional Geographer*, 28(4):371–380, 1976. 11

[71] J. Pach. Every graph admits an unambiguous bold drawing. *Journal of Graph Algorithms and Applications*, 19(1):299–312, 2015. 5

[72] D. Phan, L. Xiao, R. Yeh, P. Hanrahan, and T. Winograd. Flow map layout. In *IEEE Symposium on Information Visualization*, INFOVIS 2005, pages 219–224. IEEE, 2005. 14

[73] P. Pilehforooshha, M. Karimi, and A. Mansourian. A new model combining building block displacement and building block area reduction for resolving spatial conflicts. *Transactions in GIS*, 25(3):1366–1395, 2021. 13

[74] H.C. Purchase, R.F. Cohen, and M. James. Validating Graph Drawing Aesthetics. In *Graph Drawing: Symposium on Graph Drawing, GD '95; Passau, Germany, September 1995; Proceedings*, GD '95, pages 435–446. Springer Berlin Heidelberg, 1995. 5

[75] M.J. Roberts. *Underground Maps Unravelled: Explorations in Information Design*. Maxwell J. Roberts, 2012. 12

[76] A. Ruas. A method for building displacement in automated map generalisation. *International Journal of Geographical Information Science*, 12(8):789–803, 1998. 13

[77] K. Sahbaz and M. Basaraner. A Zonal Displacement Approach via Grid Point Weighting in Building Generalization. *ISPRS International Journal of Geo-Information*, 10(2):105, 2021. 13

[78] B. Speckmann and K. Verbeek. Homotopic C-oriented routing with few links and thick edges. *Computational Geometry*, 67:11–28, 2018. 6

[79] L.V. Stanislawski, B.P. Buttenfield, P. Bereuter, S. Savino, and C.A. Brewer. Generalisation Operators. In *Abstracting Geographic Information in a Data Rich World: Methodologies and Applications of Map Generalisation*, pages 157–195. Springer International Publishing, 2014. 13

[80] D.M. Stephen and B. Jenny. Automated layout of origin–destination flow maps: U.S. county-to-county migration 2009-2013. *Journal of Maps*, 13(1):46–55, 2017. 14

[81] M. Stilman and J.J. Kuffner. Navigation among Movable Obstacles: Real-Time Reasoning in Complex Environments. *International Journal of Humanoid Robotics*, 2(4):479–503, 2005. 7

[82] Y. Sun, Q. Guo, Y. Liu, X. Ma, and J. Weng. An Immune Genetic Algorithm to Buildings Displacement in Cartographic Generalization. *Transactions in GIS*, 20(4):585–612, 2016. 13

[83] R. Tamassia, editor. *Handbook of Graph Drawing and Visualization*. Chapman and Hall/CRC, 2013. 5

[84] A. Thomas and F. Mastrogiovanni. Minimum Displacement Motion Planning for Movable Obstacles. In *Intelligent Autonomous Systems 17*, IAS 2022, pages 155–166. Springer Nature Switzerland, 2023. 7

[85] P. Ti and Z. Li. Generation of schematic network maps with automated detection and enlargement of congested areas. *International Journal of Geographical Information Science*, 28(3):521–540, 2014. 12

[86] P. Ti, Z. Li, and Z. Xu. Automated Generation of Schematic Network Maps Adaptive to Display Sizes. *The Cartographic Journal*, 52(2):168–176, 2015. 12

[87] P. Ti, Z. Li, Z. Xu, and H. Jia. Optimizing the balance between area and orientation distortions for variable-scale maps. *ISPRS Journal of Photogrammetry and Remote Sensing*, 117:237–242, 2016. 10

[88] W. Tobler. Thirty Five Years of Computer Cartograms. *Annals of the Association of American Geographers*, 94(1):58–73, 2004. 11

[89] W.R. Tobler. CSISS: Tobler's Flow Mapper. `https://www.csiss.org/clearinghouse/FlowMapper/`. 14

[90] W.R. Tobler. Experiments In Migration Mapping By Computer. *The American Cartographer*, 14(2):155–163, 1987. 14

[91] E.R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, 2001. 2

[92] Y. Wang and M. Chi. Focus+Context Metro Maps. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2528–2535, 2011. 12

[93] J.M. Ware and C.B. Jones. Conflict Reduction in Map Generalization Using Iterative Improvement. *GeoInformatica*, 2(4):383–407, 1998. 13

[94] J.M. Ware, I.D. Wilson, J.A. Ware, and C.B. Jones. A Tabu Search Approach to Automated Map Generalisation. In *Proceedings of the 10th ACM International Symposium on Advances in Geographic Information Systems*, GIS '02, pages 101–106. Association for Computing Machinery, 2002. 13

[95] R. Weibel. Generalization of Spatial Data: Principles and Selected Algorithms. In *Algorithmic Foundations of Geographic Information Systems*, CISM School 1996, pages 99–152. Springer Berlin Heidelberg, 1997. 13

[96] G. Wilfong. Motion planning in the presence of movable obstacles. In *Proceedings of the Fourth Annual Symposium on Computational Geometry*, SCG '88, pages 279–288. Association for Computing Machinery, 1988. 7

[97] I.D. Wilson, J.M. Ware, and J.A. Ware. A Genetic Algorithm approach to cartographic map generalisation. *Computers in Industry*, 52(3):291–304, 2003. 13

[98] A. Wolff. Drawing Subway Maps: A Survey. *Informatik - Forschung und Entwicklung*, 22(1):23–44, 2007. 12

[99] H. Wu, B. Niedermann, S. Takahashi, M.J. Roberts, and M. Nöllenburg. A Survey on Transit Map Layout - from Design, Machine, and Human Perspectives. *Computer Graphics Forum*, 39(3):619–646, 2020. 12

[100] G. Xia. The Stretch Factor of the Delaunay Triangulation Is Less than 1.998. *SIAM Journal on Computing*, 42(4):1620–1659, 2013. 33, 34

[101] D. Yamamoto, S. Ozeki, and N. Takahashi. Focus+Glue+Context: An Improved Fisheye Approach for Web Map Services. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '09, pages 101–110. Association for Computing Machinery, 2009. 9, 10

[102] R. Zhao, T. Ai, and C. Wen. A Method for Generating Variable-Scale Maps for Small Displays. *ISPRS International Journal of Geo-Information*, 9(4):250, 2020. 9, 10

[103] Y. Zhou, F. Sun, W. Wang, J. Wang, and C. Zhang. Fast Updating of Delaunay Triangulation of Moving Points by Bi-cell Filtering. *Computer Graphics Forum*, 29(7):2233–2242, 2010. 38

Appendix A

Corrected divide operation of the growing algorithm

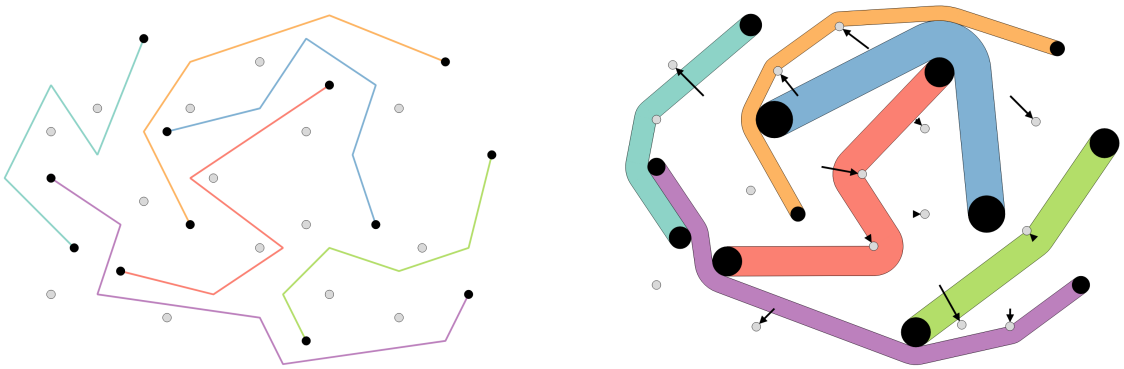
Algorithm 1: `divide` (sb, eb)

Precondition : eb, the top elbow bundle of sb, is to be merged.
Postcondition: sb_2 is the new upper straight bundle separated from sb.

```
1  if sb.right = eb then
2      sb_2 ← copy of sb
3      sb.right ← eb.inner, eb.left ← sb_2
4      sb_2.size ← eb.size, sb_2.weight ← eb.weight
5      sb.size ← sb.size − sb_2.size, sb.weight ← sb.weight − sb_2.weight
       ▷ Need to update references on left of straight
       ▷ May also need to divide one elbow
       ▷ Here we assume that sb is closer than sb_2 to their left associated vertex
       ▷ The code is similar in the other case
6      ebLeft ← sb_2.left
7      size ← ebLeft.size, weight ← ebLeft.weight
8      while size < sb_2.size do
9          ebLeft.right ← sb_2
10         ebLeft ← ebLeft.inner
11         size ← size + ebLeft.size, weight ← weight + ebLeft.weight
12     if size > sb_2.size then ▷ The current elbow bundle must be split
13         ebNew ← copy of ebLeft
14         ebLeft.inner ← ebNew
15         ebNew.size ← size − sb_2.size
16         ebNew.weight ← weight − sb_2.weight
17         ebLeft.size ← ebLeft.size − ebNew.size
18         ebLeft.weight ← ebLeft.weight − ebNew.weight
19         ebLeft.layerW ← ebNew.layerW + ebNew.weight
       ▷ The current elbow bundle is the innermost elbow bundle associated with sb_2
20     ebLeft.right ← sb_2
       ▷ Its inner elbow bundle is thus the outermost elbow bundle associated with sb
21     sb.left ← ebLeft.inner
22 else ▷ sb.left = eb
23     Repeat for this symmetrical case
```

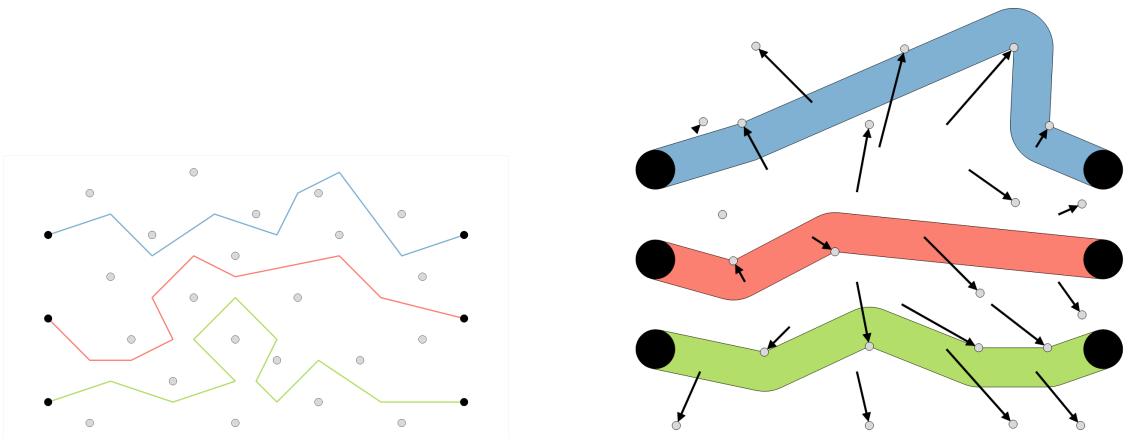
Appendix B

Experimental results



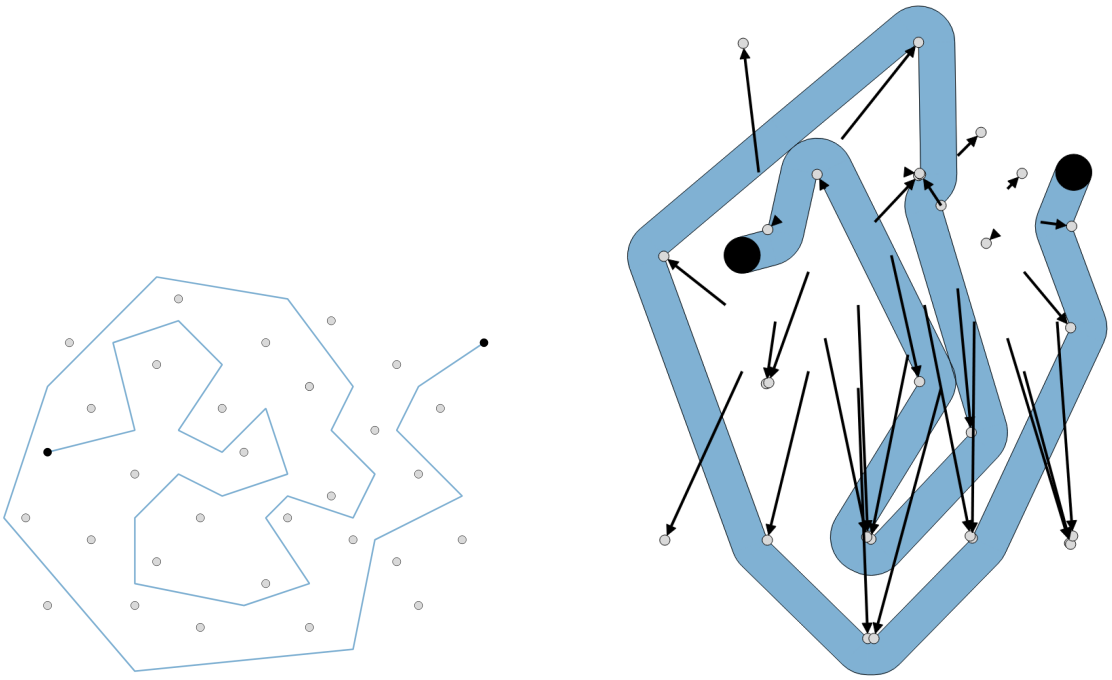
(a) Input. (b) Output.

Figure B.1: Result for instance 3.



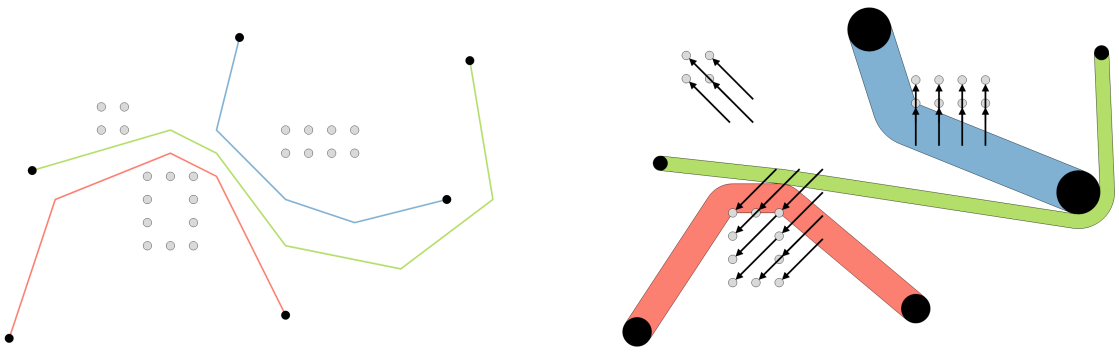
(a) Input. (b) Output.

Figure B.2: Result for instance 4.



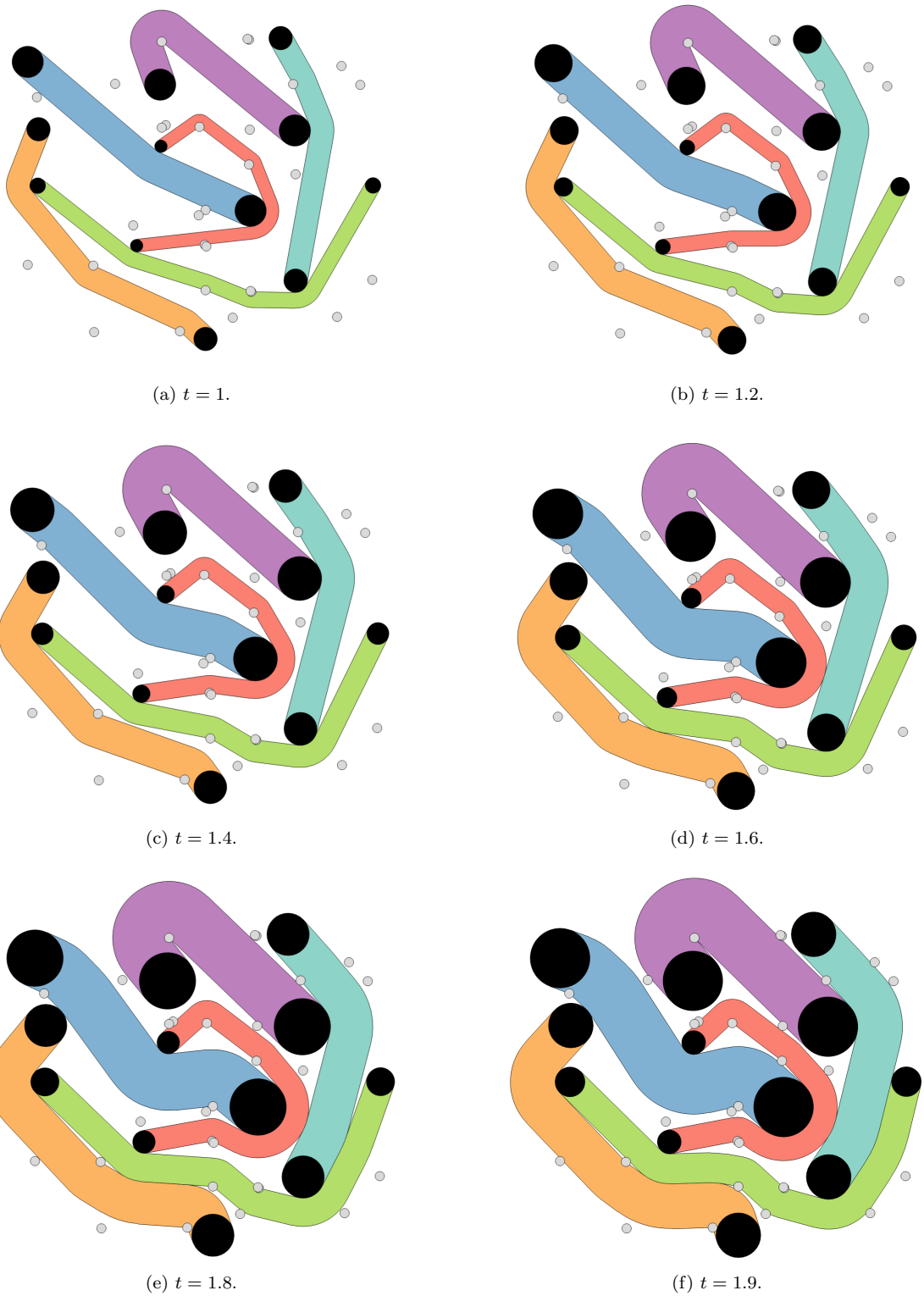
(a) Input. (b) Output.

Figure B.3: Result for instance 5.



(a) Input. (b) Output.

Figure B.4: Result for polygonal instance 2.



(a) $t = 1$.

(b) $t = 1.2$.

(c) $t = 1.4$.

(d) $t = 1.6$.

(e) $t = 1.8$.

(f) $t = 1.9$.

Figure B.5: The extended growth process of the edges for the solution in Figure 7.6(f) after reaching the required thickness at event time $t = 1$. At $t = 1.9$, the edges have reached maximum thickness proportional to their original thickness.