Visualization Algorithms for Maps and Diagrams

zur Erlangung des akademischen Grades eines

Doktor der Naturwissenschaften

der Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

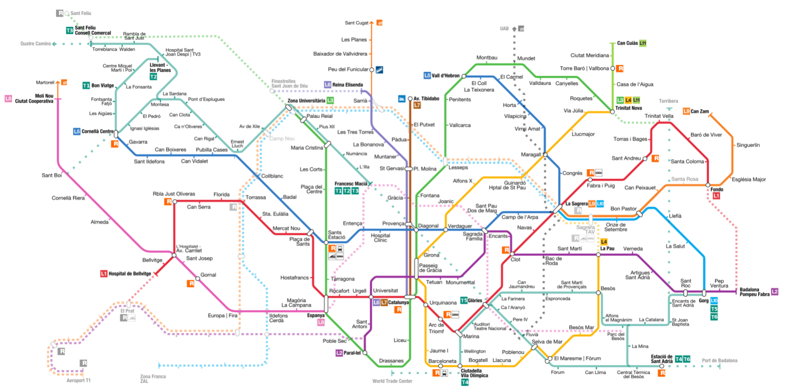
Andreas Gemsa

aus Solingen

Tag der mündlichen Prüfung:	17. Juni 2014
Erster Gutachter:	Prof. Dr. Dorothea Wagner
Zweiter Gutachter:	Prof. Dr. Bettina Speckmann
Dritter Gutachter:	Dr. Martin Nöllenburg

DEUTSCHE ZUSAMMENFASSUNG

Die Menschheit hat schon seit geraumer Zeit Karten zur Visualisierung der Umgebung genutzt. Die frühesten manuell erstellten Karten sind mehrere tausend Jahre alt, wiesen aber noch viele geographische Ungenauigkeiten auf. Durch Integration von Kartographie und Geographische Informationssysteme und anderer moderner Hilfsmittel (wie zum Beispiel Satelliten) ist das Erstellen von geographisch exakten Karten bedeutend einfacher geworden. Neben geographisch möglichst exakten Karten besteht aber auch ein großes Interesse daran, Karten zu erstellen, die es Menschen erleichtern spezielle Aufgaben durchzuführen. In solchen Karten werden bestimmte Eigenschaften hervorgehoben während andere Eigenschaften vernachlässigt werden; solche Karten geben also ein verzerrtes Bild der Realität wieder. Obwohl Straßenkarten typischerweise geographisch sehr exakt sind, wird oft die Breite der Straßen selbst übertrieben dick dargestellt. Eine besondere Art von geographischen Karten sind *schematische Karten*. Obwohl es keine allgemeine Definition von schematischen Karten gibt, so teilen die meisten schematischen Karten gewissen Eigenschaften: geometrische Details werden durch Glättung beseitigt, und Richtungen werden nur annähernd korrekt dargestellt. Typischerweise werden schematische Karten für eine spezielle Aufgabe hergestellt. Ein Paradebeispiel für solch schematische Karten sind U-Bahn Pläne. Für solche Karten kann die geographische Genauigkeit vernachlässigt werden (aber nicht vollständig) um genug Freiheit zu haben den Fokus auf die Entscheidungspunkte (hier: die U-Bahn Stationen) zu setzen. Eine solche Karte erlaubt es dem Kartenleser nun leicht eine Verbindung zwischen zwei Stationen zu finden.



U-Bahn Plan von Barcelona (`http://www.metrobarcelona.es/en/maps.html` ©Vilarubla, Creative Commons).

Neben Karten denen geographischen Daten zu Grunde liegen gibt es auch eine andere Art von Karten, oft auch *abstrakte Karten* oder manchmal *Diagramme* genannt (zum Beispiel: tree maps, oder UML Diagramme). Solche abstrakten Karten haben also keine räumlichen Informationen, die eine klare Struktur vorgeben.

Das Erstellen von Karten umfasst viele wichtige Entscheidungen, die essentiell für die Qualität des Endprodukts sind. Es gibt aber einen Schritt, der von sehr großer Wichtigkeit ist: Das Platzieren von Text und Piktogrammen auf einer Karte. Eine Karte vollständig ohne Text ist normalerweise völlig nutzlos. Für Straßenkarten sind Informationen wie Ortsnamen, Straßennamen, aber auch Piktogramme wie Symbole für Tankstellen, oder Rastplätze sehr wichtig. Dieser Schritt im Prozess des Erstellens einer Karte wird als “Positionierung von

Beschriftungen” bezeichnet (und im englischen oft als: label placement). Dass dieser Schritt in der Erstellung einer Karte große Wichtigkeit besitzt wurde vom Kartographen Imhof bemerkt: “Good name position aids map reading considerably and enhances the esthetics of the map. [...] Poor, sloppy, amateurish type placement is irresponsible; it spoils even the best image and impedes reading.” [123] (frei übersetzt etwa: Eine gute Anordnung von Namen unterstützt das Verstehen einer Karten und trägt zur ihrer Gesamtästhetik bei. [...] Schlechte und ungenaue Platzierung von Text ist unverantwortlich; es verdirbt sogar das beste Bild und behindert das Verstehen.”) Dieser Schritt ist nicht nur besonders wichtig, sondern benötigt er auch einen signifikanten Anteil des gesamten Karten Herstellungsprozesses [69]. Diese Eigenschaften zeigen, dass das automatisierte Beschriften von Karten aus algorithmischer Sicht ein interessantes Thema ist mit vielen praktischen Anwendungsmöglichkeiten.

In dieser Arbeit werden wir uns mit zwei Teilproblemen der Visualisierung von Karten und Diagrammen befassen. Zum Einen werden wir uns aus algorithmischer Sicht mit Problemen befassen bezüglich des Erzeugens von schematischen und abstrakten Karten. Zum Anderen werden wir uns mit dem Problem der Positionierung von Texten (oder Symbolen) auf Karten und Diagrammen befassen. Dabei liegt ein Hauptteil der untersuchten Problemstellungen bei dynamischen Karten. Eine dynamische Karte ist eine Karte die bestimmte kontinuierliche Operationen unterstützt (wie zum Beispiel stufenloses Vergrößern/Verkleinern oder Rotieren). Durch die wachsende Verbreitung von Smartphones in den letzten Jahren auf denen Programme dynamische Karten bereitstellen (z. B. Google Maps, iOS Maps) bieten sich hier viele neue Fragestellungen die aus algorithmischer Sicht nicht ausreichend beantwortet wurden. Die Arbeit ist nach diesen zwei Teilproblemen in zwei Hauptteile unterteilt.

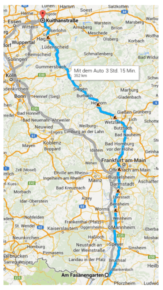
In dieser Arbeit versuchen wir immer ein Gleichgewicht zwischen Theorie und Praxis zu finden, indem wir die betrachteten Probleme erst aus Sicht der Komplexitätstheorie betrachten (z.B. durch Zeigen der NP-Schwere), und dann Algorithmen zum lösen dieser Probleme entwerfen (z.B. Approximationsalgorithmen), diese implementieren und anschließend experimentell evaluieren.

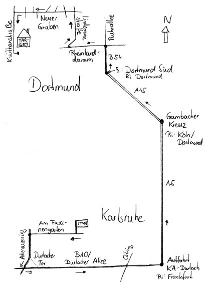
teil 1: das zeichnen von karten und diagrammen

Der erste Teil der Arbeit befasst sich mit der konkreten Zeichnung von Karten. Die erste betrachtete Problemstellung behandelt dabei räumliche Karten, während in der zweiten eine spezielle Art von abstrakten Karten betrachten wird (die Ähnlichkeit mit UML Klassendiagrammen aufweisen).

Erstellen von Routenskizzen

Das erste Problem, welches in der Arbeit betrachtet wird ist das automatisierte Erstellen von Routenskizzen. Im nebenstehenden Bild ist eine Route von Karlsruhe nach Dortmund auf Google Maps dargestellt. Allerdings ist das Bild der Route von Google Maps allein keine Hilfe, da viele Entscheidungspunkte (Abbiegungen) nicht sichtbar sind. Diese Entscheidungspunkte sind dagegen alle sichtbar auf der per Hand angefertigten Routenskizze. Da die Eigenschaften die U-Bahn Pläne auszeichnen, gut geeignet sind

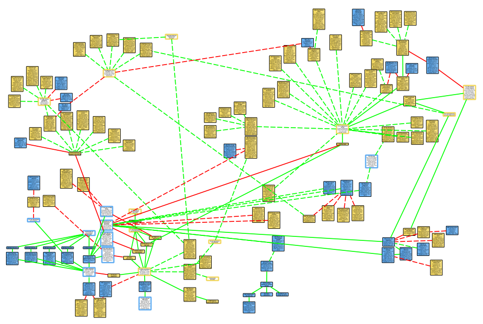




den Fokus einer schematischen Karte auf Entscheidungspunkte zu legen untersuchen wir in diesem Abschnitt der Arbeit wie man automatisiert Routenskizzen erstellen kann, die ähnlich schematisch sind wie U-Bahn Pläne, und zugleich die mentale Karte des Benutzers erhalten. Wir untersuchen das Problem zunächst aus Sicht der Komplexitätstheorie, und gehen anschließend dazu über algorithmische Ansätze zur Lösung des Problems vorzustellen und experimentell zu evaluieren. Schließlich stellen wir auch ein Webinterface für unsere Algorithmen zur Verfügung.

Zeichnen von Argumentkarten

Das zweite Problem, welches in dieser Arbeit betrachtet wird befasst sich mit der automatisierten Berechnung der Darstellung von Argumentkarten. Argumentkarten sind ein Werkzeug um die logische Struktur von Argumenten visuell darzustellen. Dabei werden Argumente in ihre Komponenten zerlegt (z.B. Prämisse, Aussage), und als Box dargestellt. Zwei Komponenten (bzw. ihre zugehörigen Boxen) sind durch einen Pfeil verbunden, wenn sie in einer Relation stehen (z.B. unterstützend). Solch eine Struktur lässt sich gut mit dem mathematischen Konzept eines gerichteten Graphen beschreiben.


Beispiel einer Argumentkarten (per Hand erzeugtes Layout).

Wir besprechen in diesem Teil der Arbeit ein algorithmisches Verfahren zum Layouten solcher Graphen. Es gibt bereits diverse Ansätze zum Zeichnen von Graphen, die von der Struktur her so sind wir die Graphen von Argumentkarten. Allerdings basieren diese meist auf einem schichtbasierten Verfahren. Wir verfolgen einen anderen Ansatz, der die Knoten des Graphs in Spalten einteilt. Das mag auf den ersten Blick kein großer Unterschied zu einem schichtbasierten Verfahren zu sein, aber bei einem schichtbasierten Verfahren ist es nicht möglich zwei Knoten in derselben Schicht miteinander zu verbinden. Das ist bei zwei Knoten in derselben Spalte sehr wohl möglich (und auch zu erwarten). Für unseren Ansatz entwickeln wir verschiedene algorithmische Lösungsstrategien, und zeigen, wenn uns möglich, NP-Schwere von einzelnen Teilproblemen. Das Kapitel wird mit einer kurzen experimentellen Analyse abgeschlossen.

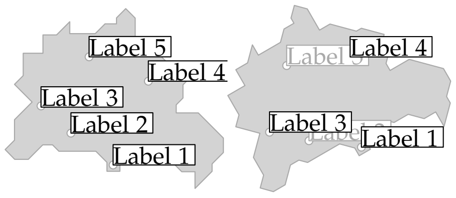
Da Argumentkarten einige Ähnlichkeiten zu UML Klassendiagrammen aufweisen besteht die Möglichkeit, dass unser Verfahren durch Modifikation auch für UML Klassendiagramme anwendbar ist. Unser Verfahren wurde in das Softwareprojekt argunet, ein Tool zum Erstellen von Argumentkarten, integriert.

TEIL 2: DAS BESCHRIFTEN VON KARTEN UND DIAGRAMMEN

Im zweiten Teil der Arbeit befassen wir uns mit dem Beschriften von Karten und Diagrammen. Dabei liegt der Fokus auf dem Beschriften von dynamischen Karten, also Karten, die kontinuierliche Operationen, wie Zoomen oder Rotieren, erlauben.

Beschriften von dynamischen Karten: Rotation

Im ersten Abschnitt betrachten wir das Problem eine Karte zu beschriften die rotiert werden kann. Wir schlagen zunächst ein Modell vor, das eine konsistente visuelle Darstellung der Beschriftungen sicherstellt. Dabei stützen wir uns auf Publikationen die sich mit dem Beschriften von dynamischen Karten, die stufenloses Zoomen erlauben, befassen. Das von uns vorgeschlagene Modell schreibt unter anderem vor, dass eine Beschriftung während einer 360° Rotation nicht mehrmals den Zustand zwischen sichtbar und unsichtbar wechseln darf, sondern maximal ein einziges mal. Diese Eigenschaft verhindert, dass die Darstellung unter viel “flackern” leidet.



Wir betrachten in diesem Kontext zwei Optimierungsprobleme und zeigen deren NP-schwere. Danach befassen wir uns mit algorithmischen Ansätzen zur Lösung der Probleme. Dabei schlagen wir für eines der Probleme umfassende algorithmische Ansätze vor: darunter fallen Heuristiken, Approximationsalgorithmen, sowie exakte Lösungsverfahren, die auf ganzzahliger linearer Programmierung basieren. Alle Verfahren wurden implementiert und wir schließen diesen Abschnitt der Arbeit mit einer umfassenden experimentellen Evaluation des vorgeschlagenen Modells, sowie der entwickelten algorithmischen Ansätze, auf real Welt Daten (extrahiert aus dem OpenStreetMap Projekt).

Beschriften von dynamischen Karten: Zoom

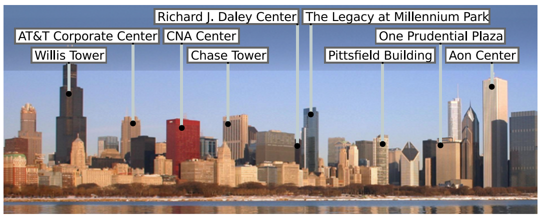
Im zweiten Abschnitt im zweiten Teil der Arbeit betrachten wir das Beschriften von dynamischen Karten die kontinuierliches vergrößern/verkleinern (zoomen) eines Kartenausschnitts zulassen. Von der geometrischen Struktur ist das Problem stark unterschiedlich zum Problem aus dem vorherigen Abschnitts. Deshalb lassen sich die dort gefundenen Ansätze *nicht* auf die Problemstellung übertragen.

Wir gehen deshalb zunächst davon aus, dass die Karte die wir beschriften wollen nur ein-dimensional ist (z.B. eine Zeitleiste). Unter dieser Annahme zeigen wir, dass sich das Problem mit Hilfe eines effizienten Algorithmus schnell lösen lässt. Wir nutzen dann Erkenntnisse aus diesem Ansatz um Approximationsalgorithmen für das Beschriften von zwei-dimensionalen Karten herzuleiten.

Beschriften von Panoramen

Das letzte Problem mit dem wir uns in dieser Arbeit beschäftigen ist das Beschriften von Panoramen. Der grundlegende Unterschied zu den bisherigen Problemen liegt darin, dass die Beschriftungen nun nicht mehr innerhalb der Karte bzw. des Diagramms liegen, sondern außerhalb platziert werden. Solch ein Ansatz ist besonders dann sinnvoll, wenn die zu beschriftenden Objekte sehr dicht aneinander liegen, oder wenn möglichst wenig Inhalt der Karte/des Diagramms durch die Beschriftungen verdeckt werden soll (sinnvoll z.B. bei medizinischen Illustrationen).

Wir betrachten das Problem des Beschriftens von Panoramabildern. Solche Bilder zeichnen sich dadurch aus, dass sie besonders breit sind. Bisher existierende Verfahren sind nicht in der Lage diese Eigenschaft sinnvoll auszunutzen. Wir untersuchen diverse Optimierungsprobleme in diesem Kontext, geben entweder effiziente Algorithmen an, oder zeigen die NP-Schwere der Probleme. Wir schließen die Untersuchung dieser Problemstellung mit einer experimentellen Analyse der vorgeschlagenen Algorithmen ab.



Beschriftetes Panorama der Skyline von Chicago. Foto: ©J. Crocker.

Computer science is no more about computers
than astronomy is about telescopes.
– unknown origins
(often misattributed to Edgar Dijkstra)

ACKNOWLEDGMENTS

Writing a PhD thesis is a lot of work and I could not have done it without many people who supported me before and during the last couple of years. Below is a brief (and incomplete) list of people to whom I owe thanks.

One of the persons without I certainly would not have finished my PhD, because I would not have started it, is Martin Nöllenburg. In 2009 he offered me a position as a scientific researcher in his newly founded Young Investigator Group. During our collaborative research he offered me always guidance and support for any of my endeavours. Thank you, Martin!

Ignaz Rutter was also of tremendous help while pursuing my PhD. Especially, at the beginning of my work as scientifc researcher he offered support and helped me immensly by showing me many helpful tips and offering advice where needed. While Ignaz claims in his thesis that this support was a welcoming distraction of the daily grind of writing his own thesis, I cannot understate how important his help was to me.

I must not forget Dorothea Wagner, who welcomed me in her group with open arms when I started in 2009. I strongly believe that the great work climate in her group is mainly due to her relaxed and just management style.

Many thanks are also due to Bettina Speckmann who not offered to let me stay and research in her own group in 2012, she also took time out of her very busy calendar and traveled from Eindhoven to Karlsruhe in a very brief period to be part of my PhD examination.

Many thanks also to Thomas and Tanja both of which were my office mates with whom I had great fun and discussed many interesting topics. Markus Völker, who is a great friend, and one of the nicest persons I've ever met. Although I really enjoy working with the group under Dorothea I especially like to thank Herr Baum, Benjamin, Franzi, Tanja, and Thomas who made working on my PhD much more than only bearable and continue to provide a fun work environment. I'd like to highlight Benjamin since he and I researched together intensively for quite a while on several interesting topics which was always a great experience. Among the people who made my life much more enjoyable is also each member of the *Escorial Committee* (each past and current member): Thomas, Markus, Moritz, and Fabian. Thanks also to Julian for our annual squash event. I also owe thanks to everyone who participated in the (usually) weekly football games and I do owe apologies to everyone who participated in the (usually) very unregularly happening 301-movie events and who suffered through one of my fantastically terrible movies.

I also need to thank Cliff, who was and still is a great friend since we met at the beginning of our studies in Karlsruhe. I'm confident he will be still a great fried even long after he and I have left Karlsruhe. Another friend I must thank is Tobias, who was always a willing victim in our FIFA matches (just kidding.. you actually play very well). Thanks are also due to Florian who remained a great friend since the early days of my childhood. Besides those three, there are many friends who were part of shaping me into the person I am today. To those of you who belong to this list: thank you guys!

I also need to thank a special person who entered my life on new years eve 2012. She was part of many new very welcomed distractions and adventures during the final years of my dissertation. Thank you Bettina.

Of course, I need to thank my parents Bernhard and Siegrun without whom I would not have had the freedom and support that made studying and subsequently starting my PhD possible. They showed several times (and I always knew) that if I needed their support they would help me, no matter what. It is impossible to explain how much that means to me and I owe them more than I can explain. An important part of becoming the person who I am today are also my siblings Sebastian and Anna. Sebastian, because he really often thinks like me and shares many of my interests, and Anna, because she really does not think like me at all, but whom I could rely on to explain me things from a different point of view. Thank you Anna, Bernhard, Sebastian and Siegrun!

INHALTSVERZEICHNIS

1

INTRODUCTION AND THESIS OUTLINE

Since prehistoric times humans have produced maps (e. g., from stone and clay [42, 184]) to convey spatial information based on their personal mental conceptualizations of their surroundings. While such old maps suffer from many geographic inaccuracies, nowadays, with the aid of modern technology (e. g., satellites, GPS) and with the integration of the map making process (also referred to as *cartography*) with geographic information science (GIS) producing maps of high geographical precision has become much easier. However, there is a demand for task specific maps that aid humans to process information much more efficiently. Such maps show a distorted representation of the spatial information, where certain features are exaggerated. Note that this is already not uncommon for road maps since there the width of the roads is usually increased beyond their true representation.

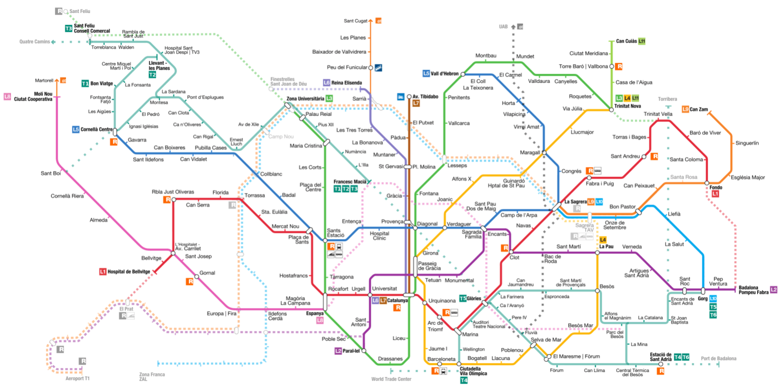


Figure 1.1: Metro map of Barcelona (http://www.metrobarcelona.es/en/maps.html ©Vilarubla, Creative Commons).

A certain kind of spatial map is called *schematic map*. Schematic maps are produced by schematizing certain spatial information to emphasize certain features of the map and de-emphasizing others. Although there is no general theory of schematization [133] schematic maps usually share the following characteristics: geometric line details are removed by smoothing and directions are only approximately preserved [8]. Usually, schematic maps are made with a certain task in mind and show *only* essential information [145]. A prime example for these kinds of maps are metro maps[1], where the goal usually is to create maps such that a reader is able to easily find a connection between two stations. For such maps, spatial correctness can be relaxed (although not completely) which allows to set the focus on the decision points of the transportation network [118].

Besides maps that depict geographical and spatial information. There are other types of maps that are non-spatial and also referred to as abstract maps or diagrams, e. g., tree maps,

1 Metro map of the London underground http://www.tfl.gov.uk/assets/downloads/standard-tube-map.pdf

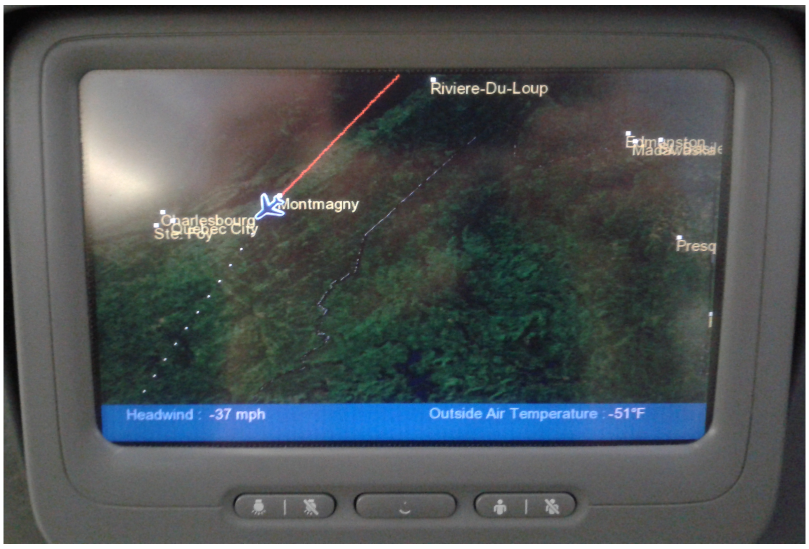


Figure 1.2: Picture of the on-board entertainment systems during a United Airlines flight. The dynamic map labeling algorithm produces multiple label overlaps.

and UML-diagrams. Although they have not necessarily any relation to geographical data they have a spatial component: "Another way of dealing with space is by use of diagrams. This is an interesting case because diagrams are intermediate to language and physical environments. A diagram is representational, inteded to convey spatial information about a place that is not physically present, just as in language. A diagram, however, is also a physical object having its own spatial properties, just as do real sketches, pictures, and so on are commonly used to provide spatial information" [43]. Contrary to the spatial maps, where, even when schematized spatial correctness can be relaxed but *not* completely omitted, for diagrams usually no spatial information is provided at all. Hence, the structure to display the information the diagram aims to provide is much less clear.

While drawing a map or a diagram entails many important decisions, one is of most critical importance to convey information: the annotation of point-, line-, or area features. Depicting a map *without* any textual information is usually of no use. For road maps this usually entails the placement of names of towns, cities, rivers, states, countries, but also placing highways symbols and other non-textual labels. This part of the map making process is usually referred to as the "placement of names on the maps" or as the "labeling of the map". This step in the map making process is of tremendous importance. As the Cartographer Imhof put it "Good name position aids map reading considerably and enhances the esthetics of the map. [...] Poor, sloppy, amateurish type placement is irresponsible; it spoils even the best image and impedes reading." [123]; see Figure 1.2 for an example. While it is not only important it also requires a significant portion of time necessary for the whole map making process [69]. This makes the problem of label placement on maps and diagrams an interesting area for algorithmic research with practical applications. The Computational Geometry Impact Task Force determined that map labeling is one of the import areas of research in Discrete Computational Geometry [53], and hence, there has been considerable amount of work done to automate the process of placing text on a map. Researchers in the field of computational geometry have been proposed algorithm to produce map labelings of high quality under certain restrictions. However, the recent popularity of hand held devices such as smart phones, which feature map applications, opened a new field of research that focuses on label placement on dynamic maps, i. e., maps that allow interactive operations such as zooming or rotation.

While the work in this thesis comes from the view of a theoretical computer scientist, we still aim to provide a balance between theoretical analysis (i. e., proving NP-hardness of investigated problems, time complexity and correctness of newly developed algorithms) and practical evaluation (i. e., implementation of the proposed algorithmic approaches and thorough experimental evaluation). In all but one of the chapters (excluding Introductions, and Preliminaries) we provide additional to the theoretical analysis an experimental evaluation. The evaluated algorithms range from heuristics, to approximation algorithms, and where possible to fast, exact algorithms. Since many of the investigated problems are NP-hard we often also include integer linear programming formulations producing exact solutions in our experimental analysis, either to take their results as benchmark, or to provide a viable alternative to non-exact algorithms, for problems with typically small problem size instances.

OVERVIEW AND CONTRIBUTION

This thesis is concerned with algorithmic approaches for visualizing maps and diagrams. The problem of visualizing maps and diagrams can be divided into two parts, and we structure this thesis along this division. In the first part of this thesis we consider certain types of maps and diagrams and how to generate a drawing that adheres to certain quality requirements. The second part of this thesis is concerned with labeling of maps and diagrams. More specifically, we consider the problem of labeling dynamic maps, that support either contiguous rotating or that support contiguous zooming. We also consider a problem which can be seen as both labeling of maps, or labeling of diagrams.

Chapter 2: Preliminaries

In this chapter, we give a very brief introduction to notation and terms that are relevant in the field of theoretical computer science and are used throughout this thesis.

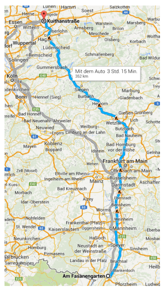
Part I: Drawing Maps and Diagrams

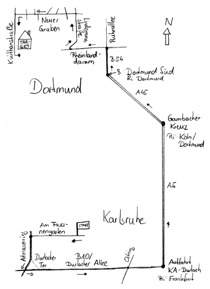
Chapter 3: Introduction

We give a brief introduction to the topic of drawing maps and diagrams, as well as a short list of related work.

Chapter 4: On d-regular Schematization of Route Sketches

Motivated from generating route sketches, or sketch maps, we consider the problem of schematizing a given shortest route in a road network. The challenge in this problem lies in the difficulty maintaining the user's mental map while emphasizing certain parts of the route and reducing the route's drawing complexity. We first show NP-hardness of the problem, give an integer linear programming (ILP) formulation for obtaining solutions, and show that under certain restrictions the problem is solvable in polynomial time with a dynamic programming algorithm. This dynamic programming algorithm is then extended towards an heuristic approach for solving the problem. We conclude this chapter of the thesis with an experimental evaluation of the proposed ILP and heuristic.

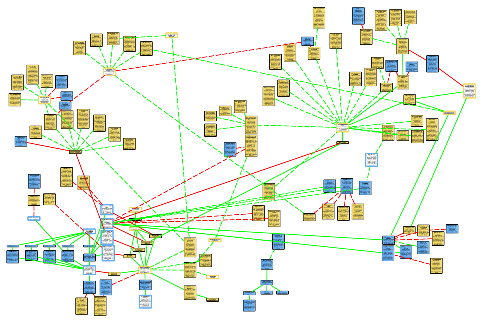




This chapter is based on joint work with Daniel Delling, Martin Nöllenburg, Thomas Pajor, and Ignaz Rutter. Preliminary results have been published in [64, 102, 63, 101].

Chapter 5: Column-based Graph Drawing

One way to structure and visualize argumental debates is to draw them as an argument map. An argument map is a diagram consisting of a box for each component of an argument, e. g., premises, objections, rebuttals. These boxes (i. e., the representation of the components of an argument) are connected by arrows when their corresponding components are in a relationship (e. g., support). Automatically generating layouts for such diagrams is a prime candidate for algorithms from the field of graph drawing.


Example of an argument map (manually created layout).

In this part of the thesis, we discuss an approach to automatically layout argument maps. Although there already exists numerous approaches for layouting graphs that are in their structure similar, our approach differs from the standard approach that would employ a layering strategy. However, we assume that the boxes have uniform width, which allows us to position the nodes into columns. We combine this approach with several newly developed algorithms to obtain an approach to compute compact layouts of argument maps. We note that argument maps share certain characteristics with UML diagrams and that our approach may also find applications there. The approach has been implemented into the software argunet which is an open source software for managing and layouting argument maps.

This chapter is based on joint work with Gregor Betz, Christof Mathies, Ignaz Rutter, and Dorothea Wagner. Preliminary results have been published in [27, 26, 71].

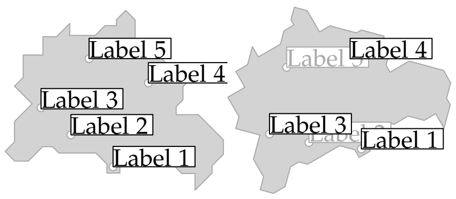
Part II: Labeling Maps and Diagrams

Chapter 6: Introduction

In this chapter we give an introduction to the problem of labeling a map. We mainly focus on models and algorithmic results concerned with labeling of point features. This part builds the basis for the following three chapters.

Chapter 7: Labeling of Rotating Maps

Rotation is one of the basic operations that most dynamic maps support. This chapter deals the problem of labeling a rotating map. We describe a model based on conistency requirements for dynamic maps proposed by Been et al. [9], and we show NP-hardness of the problem of finding an optimal labeling in this model. However, we describe an efficient polynomial-time approximation scheme based on the well-known stabbing technique by Hochbaum and Maass [117]. Further, we give two (mixed) integer linear programming formulation to obtain optimal solutions, and provide several simple heuristics. We conclude this part of the thesis with an experimental evaluation of the model and the proposed algorithmic approaches.



This chapter is based on joint work with Martin Nöllenburg, and Ignaz Rutter. Preliminary results have been published in [103, 104, 106].

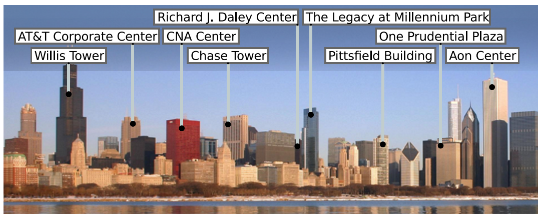
Chapter 8: Labeling of Zooming Maps with Sliding Labels

Besides Rotation, one of the other basic operations for dynamic maps is zooming. For two-dimensional maps this has already been explored by Been et al. [10], but only under the assumption that for each label there is only one possible label position relative to its point feature. We focus on labeling one-dimensional maps (i. e., lines) in the slider-model where the label can have any position relative to its point feature as long as it touches it. While we can neither show NP-hardness of the problem nor are able to give a polynomial-time algorithm, we give an algorithm that solves the problem in the k-position labeling model, where each label can have one of k specified positions relative to its point feature, optimally. We extend this algorithm to a fully polynomial-time approximation scheme for the slider model. We briefly discuss the two-dimensional problem and give simple approximation algorithms for the k-position labeling problem in the two-dimensional case.

This chapter is based on joint work with Martin Nöllenburg, and Ignaz Rutter. Preliminary results have been published in [105].

Chapter 9: Labeling Panoramas

In the last chapter of this thesis we discuss the problem of labeling panoramas. This problem differs from the previously discussed labeling problems in that we do not want to place the labels close to the point features but rather place them outside of the panorama and connect the point feature with a line (called *leader*). Existing algorithmic approaches are unsuitable because they either place the labels to the right (or left) of the panorama, but do not exploit the fact that panoramas are usually very wide. Or, they place the labels on top (or below) perpendicular to the panorama. We propose several algorithmic approaches for placing labels above (and below) the panorama such that they remain horizontally aligned, no two leader intersect, and no leader intersects a label. The chapter is concluded by an extensive evaluation of the proposed approaches.



Labeled Panorama of Chicago's skyline. Photo: ©J. Crocker.

This chapter is based on joint work with Jan-Henrik Haunert, and Martin Nöllenburg. Preliminary results have been published in [99, 98].

2 PRELIMINARIES

In this chapter we introduce basic terminology that is used throughout this work. Most of the introduced terms and notation is widely used in the field of graph theory and other fields of theoretical computer science. For a more detailed overview and description on many specific terms and approaches we refer the reader at the appropriate places to well-known literature. In the following we assume that the reader is well-versed in the basic mathematical concepts, like the big-O-notation and set theory, if not we suggest the following reading material [61, 90].

2.1 GRAPH THEORY

We begin by introducing standard terminology used in graph theory. The definitions are commonly used in the literature. For a more detailed description, as well as a broader overview of graph theory we refer the reader to [31, 67].

undirected graph An *undirected graph* G is a pair (V, E), where V is a set of vertices, and $E \subseteq V \times V$ is a set of unordered pairs of vertices, called *edges*. We denote an edge between two vertices $u, v \in V$ *adjacent* by $\{u, v\}$, and say that u and v are connected or *adjacent*. A *directed graph* $G = (V, E)$ is also *directed graph* a pair of sets, where V is a set of vertices and $E \subseteq V \times V$ is a set of *ordered* pairs, which we also refer to as *(directed) edges*. We denote a directed edge from a vertex $u \in V$ to a vertex $v \in V$ by (u, v). Further, for an edge (u, v) we call u the *source vertex* and v the *target vertex*. We usually refer to the number of vertices of an undirected/a directed graph as $n := |V|$, and to the number of edges as $m := |E|$. If, from the context, it is clear whether a graph is undirected or directed or it is unimportant, we omit the qualifier and simply call the *graph* (un)directed graph a *graph*.

path A *path* in a graph $G = (V, E)$ is a sequence $(v_1, \ldots, v_k)$ of distinct vertices $v_1, \ldots, v_k$ such that $\{v_i, v_{i+1}\} \in E$. Similarly, a *directed path* in a directed graph $G = (V, E)$ is a sequence $(v_1, \ldots, v_k)$ of distinct vertices $v_1, \ldots, v_k \in V$ such that $(v_i, v_{i+1}) \in E$.

acyclic graph We call a directed graph $G = (V, E)$ *acyclic* if there exists no vertex $v \in V$ for which there is a directed path from v to a vertex $u \in V$ with $(u, v) \in E$.

drawing We say that a *drawing* of a graph G is a function Γ that maps the vertices of G to distinct points in the two-dimensional plane, and maps the edges of G to simple Jordan curves connecting their vertices. We say a drawing of a graph is a *straight-line drawing* if each edge *planar* of G is mapped by Γ to a straight line segment. We say that a drawing is *planar* if no two edges in a the drawing intersect, except at common endpoints. We say that a graph G is *planar* if G admits a planar drawing.

The intersection graph of a set O of geometric objects in the plane is defined to be the undirected graph $G = (V, E)$ where for each $o \in O$ there is a corresponding $v_o \in V$. An edge between $v_i, v_j \in V$ exists if and only if the intersection of geometric shapes corresponding to v_i and v_j is non-empty.

2.2 ALGORITHMS AND COMPLEXITY

Although we want to find for every problem we have considered efficient algorithms, sometimes, we did not succeed in doing so. However, in those cases we were usually able to show that the considered problem is very difficult in some sense, and that it is unlikely that an efficient algorithm exists to solve it. In this section we give a short introduction to complexity theory. For a more detailed description and introduction to this important part of theoretical computer science we refer the reader to [91, 155, 61]

THE CLASSES P AND NP. Both P and NP are sets or classes of decision problems. A *decision problem* is a problem which can be answered with a *yes-or-no* answer. Contrary to decision problems there are also optimization problems. Typically, an optimization problem asks to maximize or minimize an optimization function. Those problems *do not* belong to either P or NP. There exists a wide literature regarding the complexity classes PO and NPO , which are similarly defined as P and NP but for optimization problems; see [183] for a detailed characterization of those classes.

decision problem

The class P contains all problems for which a *deterministic Turing machine* requires only polynomial time to verify its solution. We say that such problems (i. e., problems in P) can be solved *efficiently*, or that an efficient algorithm exists that solves the problem. An algorithm is considered to be *efficient* if it solves the problem, and if its running time is bounded by a function that is a polynomial of its input. In other words, an algorithm is efficient, if for input of size n the running time is in $O(n^c)$ for some fixed constant c. Note that the term efficient may be misleading and does not imply that an algorithm is useful from a practical point of view. For one, the constant c can be arbitrarily large (e. g., an algorithm with worst-case running time $\Theta(n^{1104})$ is considered to be efficient). The other reason why even an algorithm with $\Theta(n)$ worst-case running time might be irrelevant from a practical point of view is that the big-O-notation hides large constants. For an example of an efficient algorithm with large constants, we refer the reader to the publication by Robertson and Seymour [163] in which they give a $O(n^3)$ algorithm for graph minor testing. Although it seems difficult to obtain a more or less exact bound for this constant (there is a very rough estimation given in [124]) we quote David Johnson who said that: "Unfortunately, for any instance $G = (V, E)$ that one could fit into the known universe, one would easily prefer $|V|^{70}$ to even constant time, if that constant had to be one of Robertson and Seymour's" [124]. For an example of an algorithm with large exponent and large constant we refer the reader to [47][1].

The class NP contains all decision problems for which a deterministic Turing machine requires only polynomial time to verify a given solution. To understand the difference between both classes consider the problem SUBSETSUM. A SUBSETSUM instance has as input a set M of integers and asks whether there exists a non-empty subset $S \subseteq M$ whose elements add up to zero. For example: Is there a subset S for the set $M = \{-3, 4, -12, 7, 8\}$, such that the sum of all of S's elements is 0? The answer is "yes" since the elements of the set $S = \{-3, -12, 7, 8\}$ add up to 0. While determining such a solution appears to be difficult[2], verifying a solution is simple (check if the sum of all elements is 0 and whether all elements are contained in M). While it is easy to see that $\mathsf{P} \subseteq \mathsf{NP}$, it is still unknown whether $\mathsf{P} = \mathsf{NP}$ or $\mathsf{P} \neq \mathsf{NP}$. This question—also called the "P versus NP" problem—is widely considered to be one of the most interesting open problems in theoretical computer science [87].

NP-COMPLETENESS. Here, we discuss decision problems with a special property called *NP-completeness*. Consider two arbitrary decision problems Π_1 and Π_2. A *polynomial-time reduction* $\leq_p$ from Π_1 to Π_2 is a polynomial-time algorithm that computes for each instance

poly-time reduction

1 The running time of the algorithm is bounded by $1752484608000 \cdot n^{79}(\ell_{\max}/d_{\min}(\Theta_0))$.
2 there is no known algorithm that has polynomial worst-case running time

I_1 of Π_1 an instance I_2 of Π_2 such that I_2 is a *yes*-instance of Π_2 if and only if I_1 is a *yes*-instance of Π_1. This means that if we find a polynomial-time algorithm that solves the problem Π_2, then we have automatically found a polynomial-time algorithm that solves Π_1. We can also conclude that the problem Π_2 is at least as hard as Π_1.

NP-hard

NP-complete

We say that a problem Π_2 is called *NP-hard*, if for every problem $\Pi_1 \in \mathsf{NP}$ there exists a polynomial-time reduction to Π_2. Further, a problem Π_2 is called *NP-complete* if it is NP-hard and belongs to the class NP itself. The class of NP-complete problems is denoted by NPC. At this point we can infer that any problem in NP can be reduced, in polynomial time, to any problem that is in NPC. Hence, NP-complete problems can be considered the "most difficult" problems in NP and any polynomial-time algorithm for any problem in NPC directly implies that $\mathsf{P} = \mathsf{NP}$.

At this point there is still an open question. It is unclear if there are problems in NP for which there is a polynomial-time reduction from any other problem in NP. Or, more bluntly, "do NP-hard problems exist?". This was answered by Cook in his seminal paper "The Complexity of Theorem Proving Procedures"[60] in which he showed that any problem in NP can be encoded as a SAT instance of polynomial size. The SATISFIABILITY (SAT) problem asks whether a given Boolean formula in conjunctive normal form (CNF) has a variable assignment that satisfies all its clauses. Based on this result Karp [127] provided a list of 21 problems he proved to be NP-complete. For an extensive overview of NP-hard and NP-complete problems we refer the reader to [91].

weakly NP-c

pseudo poly-time

Although NP-complete problems are, in some sense, the most difficult problems in NP, we can make a distinction between two types of NP-complete problems for which the input consists of numerical parameters. We say an NP-complete problem is *weakly NP-complete* if there exists an algorithm that is polynomial in the length of the input and in the value of the numerical parameters of the input. We call such an algorithm a *pseudo polynomial-time algorithm*. At first it might seem as though the existence of such an algorithm would resolve the P vs. NP question, but integers are represented in binary, and hence require only logarithmic space with respect to their value. Hence, an algorithm that is polynomial with respect to the value of the integers of the input has exponential running time with respect to the size for storing the integers, and thus, in the size of the input. Nevertheless, pseudo polynomial-time algorithms can be used in practice if the numbers in the input are small. One problem that is only weakly NP-complete is the already mentioned SUBSETSUM problem [91]. There is a dynamic programming algorithm that solves SUBSETSUM with running time that is polynomial in the length of the input and in the values of the numerical parameters. This might imply that the inherent difficulty of weakly NP-complete problems lies in the possibility that very large numbers are permitted as input and *not* in the fundamental structure of the problem. The counter-part to weakly NP-complete problems are *strongly NP-complete* problems. An NP-complete problem is said to be strongly NP-complete if the problem remains NP-complete if the values of all numerical parameters are bounded by the length of the problems input. Hence, the existence of a pseudo polynomial-time algorithm for a strongly NP-complete problem directly implies $\mathsf{P} = \mathsf{NP}$.

strongly NP-c

Although by definition there cannot be an optimization problem that is NP-complete, there are optimization problems which are NP-hard. Such a problem can be used to determine the answer to a decision problem. For example, a valid optimization problem is to ask for a given set of integers, the maximum number of subsets whose elements add up to 0. It is obvious that an efficient algorithm which solves this problem can be used to solve the above mentioned SUBSETSUM problem.

SELECTION OF NP-COMPLETE PROBLEMS. In the following chapters of this thesis we show several NP-hardness proofs. To give a the reader a place to find all the problems we give here the formal definitions of the problems used for our reduction. We have introduced

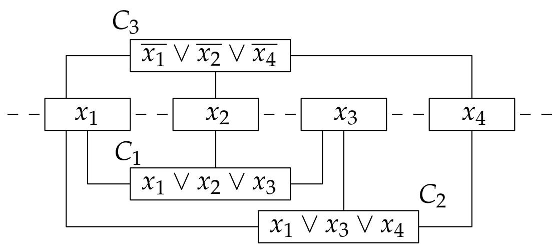


Figure 2.1: A PLANAR MONOTONE 3-SAT instance with four variables and three clauses.

several problems already in this section and will reference them when we need them later. Among the introduced problems is SAT. A somewhat "simpler" variant is 3SAT whose formal definition is as follows.

Problem: 3SAT
Instance: m clauses, n variables, in each clause at most 3 literals
Question: Is there a variable assignment that satisfies all m clauses?

This problem was shown to be NP-complete by Karp [127] and belongs to the already mentioned list of 21 NP-complete problems. Interestingly enough, the problem 2SAT whose definition is analogous to 3SAT but requires that in each clause there are at most two literals is known to be in P [137].

We discuss now a variant of 3SAT called PLANAR 3SAT. Consider any 3SAT instance $\mathcal{I}$. We can construct from $\mathcal{I}$ a graph G by adding for each variable and for each clause in $\mathcal{I}$ a vertex to G and connecting a variable-vertex with a clause-vertex if and only if the corresponding variable is contained in the corresponding clause. We call such a graph the *clause graph*. As it turns out, even if we restrict 3SAT to instances where the clause graphs is planar, the problem still remains NP-complete [139].

Problem: PLANAR 3SAT
Instance: m clauses, n variables, in each clause at most 3 variables, clause graph is planar
Question: Is there a variable assignment that satisfies all m clauses?

In fact the clause graph is not only planar, but it has a planar drawing where all variable-vertices are placed on the x-axis, and all clause variables are placed either above or below the x-axis, such that the edges connecting variable-vertices and clause vertices consist of only a vertical and a horizontal segment (the horizontal segment may have length 0). An example for this can be seen in Figure 2.1.

There are even more restricted variants of 3SAT which are still NP-complete. One of such problems is PLANAR MONOTONE 3SAT [22].

Problem: PLANAR MONOTONE 3SAT
Instance: m clauses, n variables, in each clause at most three variables, in each clause are either only positive or only negative literals, the clause graph can be drawn planar, such that the variable-vertices are placed on the x-axis and the clause-vertices containing only negative literals are placed above the x-axis and all clause-vertices containing only positive literals are placed below the x-axis.
Question: Is there a variable assignment that satisfies all m clauses?

Another well-studied NP-hard problem is the problem MAXIMUM INDEPENDENT SET. The problem is defined for a graph $G = (V, E)$ and asks for a set $V' \subseteq V$ of maximum cardinality such that for no two vertices $u, v \in V'$ there is an edge in E. For general graphs its corresponding decision problem is NP-complete [91]. Although there are approximation algorithms (e. g., [33]), no constant factor approximation algorithm for MAXIMUM INDEPENDENT SET is known. For some restricted graph classes efficient optimal algorithms are known; see e. g., [96].

Problem: Maximum Independent Set
Instance: Given a graph $G = (V, E)$.
Question: Determine a set $V' \subseteq V$ of vertices with maximum cardinality such that no two vertices in V' are adjacent in G.

MODELS OF COMPUTATION. In theoretical computer science the most commonly used model of computation is the *Random Access Machine (RAM)*. *Random Access Machine (RAM)* We omit a formal definition here and refer the reader to standard reference books in the field of computational complexity [81]. However, the reason why we are discussing this lies in the problem of using it for computations with geometry. Recall that even for problems with rational numbers as input it is easy to ask for questions where the answer is an irrational number[3]. Unfortunately, in the RAM model of computation we cannot deal with irrational numbers[4] since they have an infinite length. To avoid this problem for the field of computational geometry the *Real RAM* was proposed [39][5]. *Real RAM* The Real RAM is the same as the RAM with the exception that operations on *all* real numbers are possible. Algorithms considering geometry are often given in this model. Unfortunately, this model is not uncontroversial, since Schönhage proved that when using the Real RAM and allowing the *floor* operation it is possible to solve PSPACE problems[6] in polynomial time [167]. This, of course, requires that the designer of an algorithm heavily abuses the model. We refrain from allowing the floor operation and we will use no such tricks to improve the performance of our algorithms.

APPROXIMATION ALGORITHMS. Although we cannot expect to find a polynomial-time algorithm for an NP-complete problem, unless $\mathsf{P} = \mathsf{NP}$, we are usually still interested in finding solutions to optimization problems. There are different ways to cope with the difficulty of NP-hard problems. For some problems it might be feasible to do an exhaustive search on the solution space, or to use more advanced techniques. However, often, especially if computational time or power is limited, using such approaches is infeasible. An alternative is to find algorithms with polynomial-time running time, which do not necessarily find the optimal solution to a problem instance but for which it is possible to prove a quality guarantee. More specifically, let Π be a maximization problem and let OPT denote the optimal solution for a given instance of Π. Then, we say an algorithm $\mathcal{A}$ is a *ρ-approximation algorithm* *ρ-approximation algorithm* if for every instance I of Π the result of $\mathcal{A}(I)$ is within ρOPT of OPT, i. e., $\rho\text{OPT} \leq \mathcal{A}(I) \leq \text{OPT}$. The analogous definition holds for approximation algorithms for minimization problems, only that for $\mathcal{A}(I)$ the equation $\text{OPT} \leq \mathcal{A}(I) \leq \rho\text{OPT}$ holds. *approximation ratio* We call ρ the *approximation ratio* or *approximation factor*.

In this thesis we also discuss a certain, special type of approximation algorithms called *polynomial-time approximation scheme* (*PTAS*). *PTAS* A PTAS is an approximation algorithm that takes, additionally to the problem instance, another parameter $\varepsilon > 0$. The PTAS computes then a solution to the problem instance that is within $(1 - \varepsilon)$OPT for maximization problems and within $(1 + \varepsilon)$OPT for minimization problems. The running time of a PTAS is polynomial in n for every fixed ε. From a practical point of view, however, a PTAS might still require an prohibitive amount of computational power. It is possible to be a bit more fine-grained with respect to the time complexity of a PTAS. PTASes with a running time that is polynomial in $O(n^c)$ for a constant c that is independent of ε are called *efficient polynomial-time approximation schemes* (*EPTAS*). *EPTAS* An even more restrictive definition applies to *fully polynomial-time approximation schemes* (*FPTAS*). *FPTAS* Every FPTAS is a PTAS, but the running

3 Q: What is the diameter of a unit square? A: $\sqrt{2}$.
4 There are exceptions (e. g., when the irrational numbers are all algebraic we can store them in finite space and can operate on the numbers in finite time or when dealing with irrational numbers is unnecessary [51])
5 sometimes it is also referred to as the Blum-Shub-Smale machine which is an equivalent model [29]
6 PSPACE is the complexity class that contains all decision problems which can be solved with polynomial space in the size of the input. This implies that PSPACE is a superset of NP.

time is required to be polynomial in n and $1/\varepsilon$.

We want to mention another form of approximation scheme, the *quasi-polynomial time approximation scheme (QPTAS)*. A QPTAS is an approximation-scheme with *quasi-polynomial* time complexity. An algorithm with quasi-polynomial time complexity has a time complexity that is larger than a polynomial, but not quite large enough to have exponential time complexity. An algorithm with quasi-polynomial time complexity may have time complexity $2^{O((\log n)^c)}$, for some constant c. Hence, a QPTAS has time complexity $O(n^{O(\text{polylog}(n))})$ for every fixed ε.

QPTAS

quasi-polynomial

For an more detailed overview of approximation algorithms, as well as some standard techniques for constructing such algorithms, we refer the reader to [179].

2.3 LINEAR PROGRAMMING

A general tool that allows to formulate and solve a problem is linear programming, which is a mathematical method for solving optimization problems. Linear programs can be expressed in the so called *canonical form*. A linear program consists of a linear objective function, and a set of constraints which are either linear equality constraints or linear inequality constraints.

ALGORITHMS. The most well-known and in practice very well-performing algorithm to solve linear programs is the *simplex algorithm* by Dantzig [62]. Unfortunately, the algorithm is *not* a polynomial-time algorithm as there are instances which require the algorithm to take an exponential number of steps [132]. The first algorithm that broke the polynomial-time barrier was the *ellipsoid algorithm* given by Shor [170]. However, it was Haĉijan who showed that this approach has a polynomial-time complexity for linear programs [109]. More specifically, this algorithm has a running time of $O(n^4 L)$, where n is the number of variables and L is the length of the input encoded in bits. Unfortunately, this algorithm performs usually worse than the simplex algorithm in practice [52]. A combination of an algorithm that performs well in practice and has a good theoretical worst-case time complexity was proposed by Karmarkar [126]. The algorithm by Karmarkar has a worst-case time complexity of $O(n^{3.5} L)$.

simplex algorithm

ellipsoid algorithm

Note that the above mentioned polynomial-time algorithms are only weakly polynomial-time algorithms, i. e., their running time depends on the values of the input numbers. Or, in other words, there are no known linear programming algorithms that are strongly-polynomial in general. This particularly means that there is no known polynomial-time algorithm that solves linear programs where the coefficients are irrational numbers. There are, however, exceptions: Should the coefficients be in $\mathbb{R}$ but *algebraic numbers*, then there is an algorithm that is based on the ellipsoid algorithm which can solve such linear programs in polynomial time [17]. We want to remind the reader that an algebraic number is a number that is a root of a non-zero polynomial with only rational coefficients. Numbers which are not algebraic are called *transcendental numbers*.

For some decision or optimization problems a formulation as linear program is only possible with the additional requirement that all variables have only integer values. This is known as *integer linear programming* and was shown to be NP-complete [127][7]. However, there are commercial ILP solvers[8] that can be used to solve integer linear programs in a reasonable amount of time. This approach can be used to obtain a solution of instances of NP-hard problems.

integer linear programming

7 In this paper Karp proves that this problem is already NP-hard if all variables are binary, i. e., either 0 or 1.

8 In this thesis we give ILP formulation for several problems. For those ILPs we evaluate experimentally we use the commercial ILP solver Gurobi (http://www.gurobi.com/)

Part I

DRAWING MAPS AND DIAGRAMS

3

INTRODUCTION – DRAWING MAPS AND DIAGRAMS

In the first part of this thesis we discuss two approaches to draw maps and diagrams. The first approach deals with spatial information, and the second approach considers a special class of non-spatial maps, called *argument maps*. Since both problems use very different approaches we omit a detailed description of related work here and report it only in their respective chapters. Here, we give only a brief summary on the problems discussed in the following two chapters.

For the first topic which is described in Chapter 4 we consider the problem of schematizing a single route in a road network (e. g., as a supplement to the output of a shortest path query by a service such as Google or Bing Maps). Consider the example in Figure 3.1 that depicts a route from Bremen to a destination in Cuxhaven. This picture alone is of no use for a person trying to drive from Bremen to this destination in Cuxhaven as many important decision points along the way cannot be seen since the scale of the map is too small. Since the visual representation of metro maps is well-known among the general population we investigate how to automatically generate route sketches that resemble in their appearance the layout used for metro maps [153]. This enables us to generate a visualization of a route where the focus is on its decision points. However, in comparison to metro maps we need to be more strict with respect to the position of the decision points in the visualization to ensure that the user's mental map is maintained. More specifically, we require that the left-right/top-bottom relationship between any two decision points remains unchanged. This is not necessarily the case for metro maps, where the left-right/top-bottom relationship between any two non-adjacent stations is usually of less importance. We consider the computational complexity of the underlying problem (and show NP-hardness), propose several algorithmic approaches to solve the problem (although not necessarily optimal), and conclude the chapter with an experimental evaluation on real-world data. We have also implemented a web interface for our algorithm to allow an easy access to the functionality of the algorithms; see Figure 3.2.

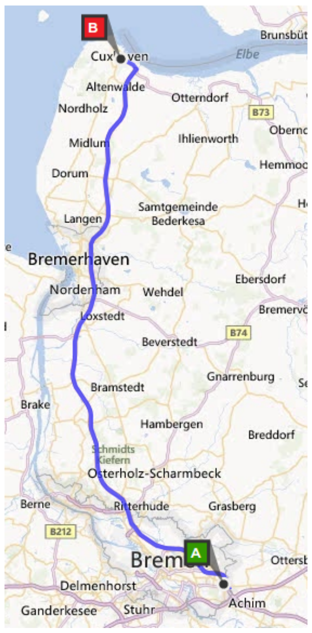
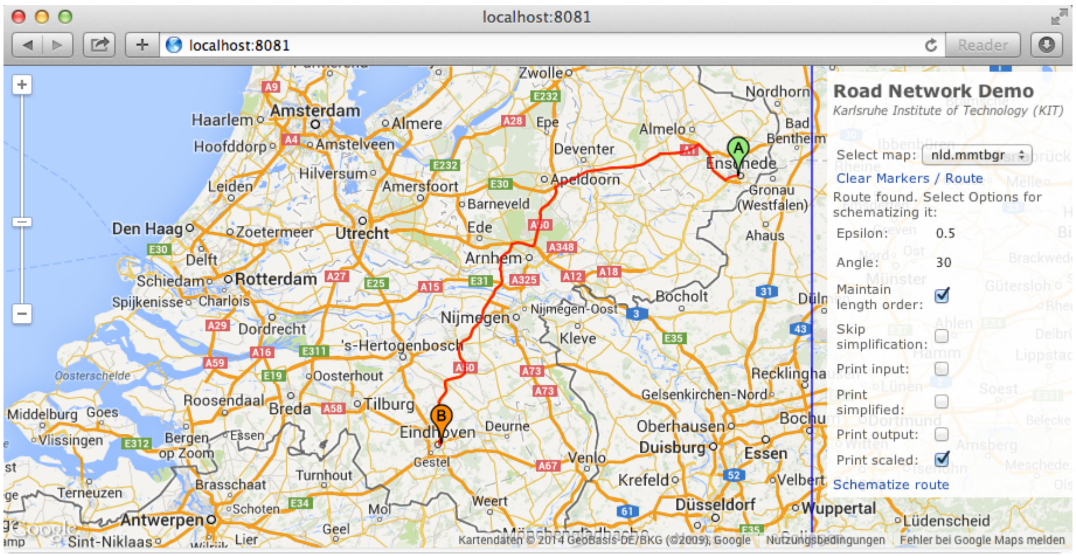


Figure 3.1: Route from Bremen to Cuxhaven (Bing Maps).

The second problem we consider is the drawing of argument maps. Argument maps are used to visualize arguments in, e. g., a debate, which then allows user afterwards, or in real-time, to analyze the structure of the whole argumentation and their relationships. An argument comprises of typically of several components (e. g., premises, objects). Each component of an argument is usually represented in an argument map by a box, and two such boxes are connected if their corresponding components are in a relationship (e. g., support); see Figure 3.3. The underlying structure of an argument map can be captured by the concept of a graph. More specifically,



Road Network Demo
Karlsruhe Institute of Technology (KIT)

Select map: nld.mmtbgr
Clear Markers / Route
Route found. Select Options for schematizing it:

Epsilon:	0.5
Angle:	30
Maintain length order:	✓
Skip simplification:	☐
Print input:	☐
Print simplified:	☐
Print output:	☐
Print scaled:	✓

Schematize route

Figure 3.2: Web interface. Visualization uses Google Maps. The route from "A" to "B" is from Enschede to Eindhoven.

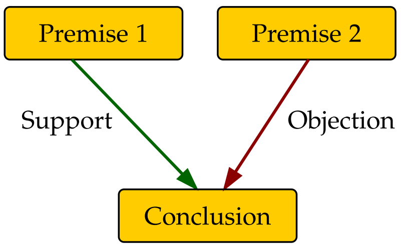


Figure 3.3: Example of a visual representation of a simple argument map.

an argument map can be seen as a *directed* graph and we can even assume that the graph is acyclic, since the structure of an argument should (usually) not contain any cycles.

For drawing hierarchical (directed and acyclic) graphs there have been many different approaches considered. However, they mainly focus on producing *layered* drawings, with, e. g., the Sugiyama framework [175]. The Sugiyama framework consists of three individual steps, which are executed sequentially. In the first step, the algorithm assigns each node to a layer, then—to reduce the number of crossings—a node ordering within each layer is computed, and as a last step, the final coordinates are computed. We propose a completely different approach towards drawing hierarchical graphs. For this we assume that all nodes have the same width, but may have individual heights. This allows us to divide the plane into equally sized columns, and position each node within a column. Note that at first glance this seems like the Sugiyama approach and we only switch layers with columns. However, the main difference lies in that in the Sugiyama framework, nodes inside the same layer *cannot* be connected by an edge, while nodes in the same column *can* (and most often *are*) connected by an edge. We propose an approach that follows the topology-shape-metric framework proposed by Tamassia [177]. We look at each step of the framework separately, consider the computational complexity (although we are not always successful in determining if the problem considered in the step is NP-hard), and give algorithms that can be used in practice to solve the problem (not always optimally). We conclude the chapter with an experimental evaluation of real-world argument maps. Our approach has been integrated in the argument map drawing software argunet[1].

1 `http://www.argunet.org/`

4

PATH SCHEMATIZATION

Motivated by drawing route sketches, we consider the d-regular path schematization problem. For this problem we are given an embedded path P (e.g., a route in a road network) and a positive integer d. The goal is to find a *d-schematized* embedding of P in which the orthogonal order of all vertices in the input is preserved and in which every edge has a direction that is an integer multiple of $(90/d)^\circ$. We show that deciding whether a path can be d-schematized is NP-complete for any positive integer d.

Despite the NP-hardness of the problem we still want to be able to generate route sketches and thus need to solve the d-regular path schematization problem. We explore two different algorithmic approaches, both of which consider two additional quality constraints: We require that every edge is drawn with a user-specified minimum length and we want to maximize the number of edges that are drawn with their preferred direction. The first algorithmic approach restricts the input paths to be axis-monotone. Although there is already an efficient algorithm that solves this problem [97] but it has a time complexity of $O(n^5 \log n)$. We propose a different, much simpler algorithm, that has a time complexity of only $O(n^2)$. Further, we extend this approach by a heuristic such that it can handle arbitrary simple paths (this also has been done before [97] but with worse time complexity). However, for the second step we cannot guarantee that the orthogonal order of the input embedding is maintained. The second approach is a formulation of the d-regular path schematization problem as a mixed-integer linear program. Finally, we give an experimental evaluation which shows that both approaches generate reasonable route sketches for real-world data.

This chapter is based on joint work with Daniel Delling, Martin Nöllenburg, Thomas Pajor, and Ignaz Rutter. Preliminary results have been published in [64, 102, 63, 101].

4.1 INTRODUCTION

Angular or $\mathcal{C}$-oriented schematizations of graphs refer to a class of graph drawings, in which the admissible edge directions are limited to a given set $\mathcal{C}$ of (usually evenly spaced) directions. This includes the well-known class of orthogonal drawings and extends more generally to k-linear drawings, e.g., octilinear metro maps. Applications of schematic drawings can be found in various domains such as cartography, VLSI layout, and information visualization.

In many schematization scenarios the input is not just a graph but a graph with an initial drawing that has to be schematized according to the given set of directions. This is the case, e.g., in cartography, where the geographic positions of network vertices and edges are given [46], in sketch-based graph drawing, where a sketch of a drawing is given and the task is to improve or schematize that sketch [34], or in dynamic graph drawing, where each drawing in a sequence of drawings must be similar to its predecessor [38]. For such a redrawing task it is crucial that the mental map [144] of the user is preserved, i.e., the output drawing must be as similar as possible to the input. Misue et al. [144] suggested preserving the *orthogonal order* of the input drawing as a simple criterion for maintaining a set of basic spatial properties of the input, namely the relative above/below and left/right positions of all pairs of input vertices. The orthogonal order has been used successfully as a means for maintaining the mental map [113, 73, 36, 63].

The motivation behind the work presented here is the visualization of routes in road networks as sketches for driving directions. An important property of a route sketch is that it focuses on road changes and important landmarks rather than exact geography and distances. Typically the start and destination lie in populated areas that are locally reached via a sequence of relatively short road segments. On the other hand, the majority of the route typically consists of long highway segments with no or only few road changes. This property makes it difficult to display driving directions for the whole route in a single fixed-scale map since some areas require much larger scales than others in order to be readable. The strength of route sketches for this purpose is that they are not drawn to scale but rather use space proportionally to the route's complexity.

RELATED WORK. Geometrically, we can consider a route to be an embedded path in the plane. The simplification of paths (or polylines) in cartography is well studied and the classic line simplification algorithm by Douglas and Peucker [72] is one of the most popular methods. Two more recent algorithms were proposed for $\mathcal{C}$-oriented line simplification [143, 148]. These line simplification algorithms, however, are not well suited for drawing route sketches since they keep the positions of the input points fixed or within small local regions around the input points and thus edge lengths are more or less fixed. On the other hand, Agrawala and Stolte [3] presented a system called *LineDrive* that uses heuristic methods based on simulated annealing to draw route sketches. A problem similar to drawing route sketches is generating destination maps, where given a destination (e.g. a restaurant) the aim is to produce a schematized map that helps anyone within a given region around the destination to reach it. Kopf et al. consider this problem [134] and give a a heuristic algorithm based on the continuation method. The methods of both of these approaches allow distortion of edge lengths and angles. However, they do not restrict the set of edge directions and do not give hard quality guarantees for the mental map such as the preservation of the orthogonal order.

A graph drawing problem that has similar constraints as drawing route sketches is the metro-map layout problem, in which an embedded graph is to be redrawn octilinearly. The problem is known to be NP-hard [150] but it can be solved successfully in practice by mixed-integer linear programming [153]. The existing methods, covered in a survey by Wolff [185], do aim to keep the mental map of the input, but no strict criterion like the orthogonal order is applied. Brandes and Pampel [36] studied the path schematization problem in the presence of orthogonal order constraints in order to preserve the mental map. They showed that deciding whether a rectilinear schematization exists that preserves the orthogonal order of the input path is NP-hard. They also showed that schematizing a path using arbitrarily oriented unit-length edges is NP-hard. Recently, the authors have published more simplified version of both proofs [37]. In [173] Speckmann and Verbeek consider the problem of finding a rectilinear, homotopic schematization of a collection of paths with a minimum number of path segments. The problem turns out to be NP-hard, but the authors give an approximation algorithm. Verbeek extends the approach in [180] to $\mathcal{C}$-oriented paths, where the path segments in the schematization can have only orientations that are contained in $\mathcal{C}$. However, besides the homotopy requirement, no special emphasis is placed on maintaining the user's mental map. Another type of $\mathcal{C}$-oriented schematization that has applications in generating schematized maps is studied by Buchin et al. [45]. There, the authors give an algorithm that finds a $\mathcal{C}$-oriented schematization of a polygon, where the polygon's edges must have directions that are contained in $\mathcal{C}$ and where the schematized polygon preserves the area of the input polygon.

contributions. Motivated by drawing route sketches, we investigate the problem of generating $\mathcal{C}$-oriented orthogonal-order preserving drawings. In this chapter we prove that deciding whether a $\mathcal{C}$-oriented orthogonal-order preserving drawing of an embedded input path exists is NP-complete, even if the path is simple. This is true for every *d-regular* set $\mathcal{C}$ of directions that have angles that are integer multiples of $90°/d$ for any integer d. This extends the result of Brandes and Pampel [36, 37] who showed NP-completeness for the case $d = 1$. However, their proof relies on the absence of diagonal edges and hence does not extend to larger values of d. We show the NP-completeness in the octilinear case $d = 2$ in Section 4.3 and, subsequently in Section 4.4, how this result extends to the general d-regular case for $d > 2$. In Section 4.5 we show that if we restrict the input to *axis-monotone* paths, i.e., x- or y-monotone paths, the d-regular path schematization problem can be solved in polynomial time. We present an efficient algorithm that finds, for a given embedded axis-monotone path, an embedding that uses only d-regular directions and maintains the orthogonal order of the input path. The algorithm maximizes the number of edges that are embedded with their preferred direction, i. e., the d-regular direction that is closest to the direction in the input embedding. We also heuristically extend this approach to non-monotone paths. In Section 4.6 we design a mixed integer-linear program (MIP) for solving the d-regular path schematization problem for arbitrary paths. Finally, in Section 4.7 we give an experimental evaluation of both the MIP approach and the heuristic simple path schematization algorithm. The source code of our proof-of-concept implementation in C++ is made available[1]. We summarize our results in Section 4.8.

axis-monotone

4.2 preliminaries

plane embedding

A *plane embedding* of a graph $G = (V, E)$ is a mapping that maps every vertex $v \in V$ to a distinct point $\pi(v) = (x_\pi(v), y_\pi(v))$ and every edge $e = uv \in E$ to the line segment $\pi(e) = \overline{\pi(u)\pi(v)}$ such that no two edges e_1, e_2 cross in π except at common endpoints. For simplicity we also use the terms vertex and edge to refer to their images under an embedding, and, unless stated otherwise, we assume every embedding to be a plane embedding. We denote the length of an edge e in π as $|\pi(e)|$.

orthogonal order

Let π and ρ be two embeddings of the same graph G. We say that ρ preserves the *orthogonal order* [36] of π if for any two vertices u and $v \in V$ it holds that $x_\pi(u) \leq x_\pi(v) \Rightarrow x_\rho(u) \leq x_\rho(v)$ and $y_\pi(u) \leq y_\pi(v) \Rightarrow y_\rho(u) \leq y_\rho(v)$. In other words, the orthogonal order defines the relative above-below and left-right positions of any two vertices.

$\mathcal{C}$-oriented

d-regular

For an input embedding π we measure the direction $\alpha_\pi(uv)$ of an edge uv as the counterclockwise angle formed between the horizontal line through $\pi(u)$ and the line segment $\pi(uv)$. For a set of angles $\mathcal{C}$ we say that a drawing is *$\mathcal{C}$-oriented* if the direction of every edge $e \in E$ is contained in $\mathcal{C}$. A set of directions $\mathcal{C}$ is called *d-regular* for an integer d if $\mathcal{C} = \mathcal{C}_d = \{(i \cdot 90°/d) \bmod 360 \mid i \in \mathbb{Z}\}$. Note that it appears that the orientation of an edge is for our problem irrelevant with the implication that the direction 45° and 225° are in principle identical. This is true for the NP-completeness proofs in Sections 4.3 and 4.4, but for our efficient algorithm for schematizing axis-monotone paths (see Section 4.5) this is different. Assuming the axis-monotone path is x-monotone, then the set $\mathcal{C}$ can contain arbitrary directions, as long as it also contains $0°, 90°, 270°$ as well as at least one diagonal direction strictly between 0° and 90° and at least one diagonal direction strictly between 270° and 360°.

d-schematization

Let (G, π) denote a graph G with a plane input embedding π. For brevity we may refer to the tuple (G, π) as (embedded) graph. A *d-regular schematization* (or *d-schematization*) of (G, π) is a plane embedding ρ that is $\mathcal{C}_d$-oriented, that preserves the orthogonal order of π

1 http://i11www.iti.kit.edu/_media/projects/routesketch.zip

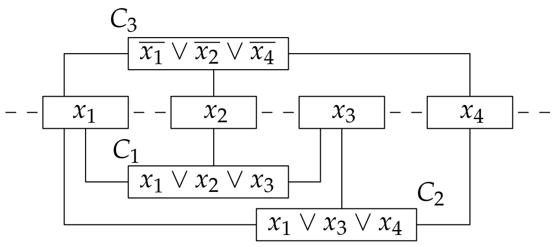


Figure 4.1: A PLANAR MONOTONE 3-SAT instance with four variables and three clauses.

and where no two vertices are embedded at the same coordinates. We also call ρ *valid* if it is a d-schematization of (G, π).

We denote by $\omega_{\mathcal{C}}(e)$ the *preferred direction* $\gamma \in \mathcal{C}$ of an edge e, i. e., the direction in $\mathcal{C}$ that is closest to $\alpha_\pi(e)$. For a $\mathcal{C}$-oriented embedding ρ of P and an edge $e \in P$ there is a *direction cost* $c_\rho(e)$ that captures by how much the direction $\alpha_\rho(e)$ deviates from $\omega_{\mathcal{C}}(e)$. More specifically, we say the direction cost of an edge is 1 if it is not embedded with its preferred direction and 0 otherwise. Note that there are edges which need to be embedded horizontally (vertically) to respect the orthogonal order of the input. The *schematization cost* $c(\rho)$ is then defined as $c(\rho) = \sum_{e \in P} c_\rho(e)$.

preferred direction

direction cost

schematization cost

Finally, a *monotone path decomposition* for a given embedded path (P, π) is a decomposition $\mathcal{P} = \{(P_1, \pi_1), \ldots, (P_k, \pi_k)\}$ of (P, π) into consecutive axis-monotone embedded subpaths (P_i, π_i) for $1 \le i \le k$, where π_i equals π restricted to the subpath P_i.

4.3 NP-COMPLETENESS OF 2-REGULAR PATH SCHEMATIZATION

In this section we show that the problem of deciding whether there is a 2-schematization for a given embedded graph (G, π) is NP-complete, even if G is a simple path. In the latter case we denote the problem as the (d-regular) *Path Schematization Problem* (PSP). We fix $d = 2$. To prove that 2-regular PSP is NP-complete we first show hardness of the closely related 2-regular Union of Paths Schematization Problem (UPSP), where (G, π) is a set $\mathcal{P}$ of k disjoint embedded paths $\mathcal{P} = \{P_1, \ldots, P_k\}$. We then extend this proof to a single path and conclude with a proof that the problem is in NP.

We show that 2-regular UPSP is NP-hard by a reduction from PLANAR MONOTONE 3-SAT, which is known to be NP-hard [22]. PLANAR MONOTONE 3-SAT is a special variant of PLANAR 3-SAT where each clause either contains exactly three positive literals or exactly three negative literals and additionally, the variable-clause graph admits a planar drawing such that all variables are on the x-axis, the positive clauses are embedded below the x-axis and the negative clauses above the x-axis. An example instance of PLANAR MONOTONE 3-SAT is depicted in Figure 4.1. In a second step, we show how to augment the set of paths $\mathcal{P}$ to form a single simple path P that has the same properties as $\mathcal{P}$ and thus proves that PSP is NP-hard. This two-step approach is similar to the way Brandes and Pampel proved NP-hardness for the case $d = 1$ [36]. They also first proved NP-hardness for a set of paths and then extended the proof to show hardness for a union of paths. However, it seems highly unlikely that it is possible to adapt the gadgets in their proof (neither from the original proof [36] nor from the simplified version [37]) to our version of the problem. Their gadgets rely on being able to force with the embedding of a single edge another edge to be embedded either vertically or horizontally. This seems very difficult to achieve when allowing diagonal directions.

In the following, we assume that φ is a given PLANAR MONOTONE 3-SAT instance with variables $X = \{x_1, \ldots, x_n\}$ and clauses $C = \{c_1, \ldots, c_m\}$.

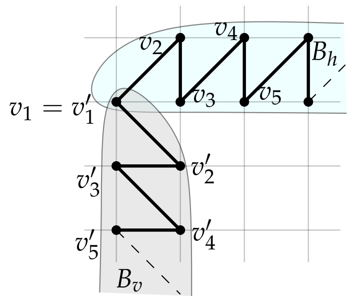


Figure 4.2: Border gadget B that is rigid and rational. The horizontal component is denoted by B_h and the vertical one by B_v.

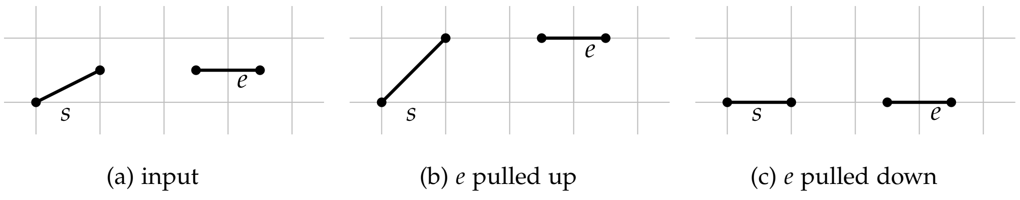


Figure 4.3: The switch s is linked with edge e; (a) shows the input embedding, in (b) e is pulled up and in (c) it is pulled down.

4.3.1 *Hardness of the Union of Paths Schematization Problem*

We begin by introducing the different types of required gadgets and then show how to combine them in order to prove the hardness of UPSP.

BORDER GADGET. For all our gadgets we need to control the placement of vertices on discrete positions. Thus our first gadget is a path whose embedding is unique up to scaling and translation. This path will induce a grid on which we subsequently place the remaining gadgets.

Let $\Delta_x(u, v)$ denote the x-distance between the two vertices u and v. Likewise, let $\Delta_y(u, v)$ be the y-distance between u and v. We call a simple path P with embedding π *rigid* if a 2-schematization of (P, π) is unique up to scaling and translations. Further, we call an embedding π of P *rational* if there exists a length $\ell > 0$ such that for any two vertices u, v of P it holds that $\Delta_x(u, v) = z_x \cdot \ell$ and $\Delta_y(u, v) = z_y \cdot \ell$ for some $z_x, z_y \in \mathbb{Z}$. Thus in a rational embedding of P all vertices are embedded on a grid whose cells have side length ℓ. For our border gadget we construct a simple path B of appropriate length with embedding π that is both rigid and rational. Hence, π is essentially the unique 2-schematization of (B, π), and after rescaling we can assume that all points of B lie on an integer grid. The border gadget consists of a horizontal component B_h and a vertical component B_v which share a common starting vertex $v_1 = v'_1$; see Figure 4.2. The component B_h alternates between a 45° edge to the upper right and a vertical edge downwards. The vertices are placed such that their y-coordinates alternate and hence all odd, and all even, vertices have the same y-coordinate, respectively. The vertical component consists of a copy of the horizontal component rotated by 90° in clockwise direction around v_1. To form B, we connect B_v and B_h by identifying their starting points v_1 and v'_1.

rigid

rational

Lemma 4.1 *The border gadget B with its given embedding π is rigid and rational.*

Proof. First note that in B_h due to the orthogonal order all vertices v_i with i odd and all vertices v_i with i even have the same y-coordinates, respectively, in any valid schematization. Hence $\Delta_y(v_i, v_{i+1})$ is the same for all i. We show that v_i and v_{i+1} for i odd, also have the same x-distance $\Delta_x(v_i, v_{i+1}) = \Delta_y(v_i, v_{i+1})$.

Since the edge v_1v_2 connects two vertices with different y-coordinates and since v_2v_3 must be embedded vertically, v_1v_2 must be embedded with an angle of 45°. And hence we have $\Delta_x(v_1, v_2) = \Delta_y(v_1, v_2)$. The same argument holds for all edges v_iv_{i+1} with i odd and hence B_h is both rational and rigid.

Since the vertical border gadget B_v is a copy of B_h it is also rational and rigid. Moreover, we have that $\Delta_x(v'_1, v'_2) = \Delta_x(v_1, v_2)$, i.e., the distances between vertices of B_v that lie on the same horizontal or vertical lines are the same as for B_h. Hence the border gadget B is rigid and rational. □

In the following we define the length of a grid cell induced by B as 1 so that we obtain an integer grid. We choose B long enough to guarantee that any vertex of our subsequent gadgets lies vertically and horizontally between two pairs of vertices in B.

The grid in combination with the orthogonal order gives us the following properties:

1. If we place a vertex v with two integer coordinates, i.e., on a grid point, its position in any valid embedding is unique;
2. if we place a vertex v with one integer and one non-integer coordinate, i.e., on a grid edge, its position in any valid embedding is on that grid edge;
3. if we place a vertex v with two non-integer coordinates, i.e., in the interior of a grid cell, then its position in any valid embedding is in that grid cell (including its boundary).

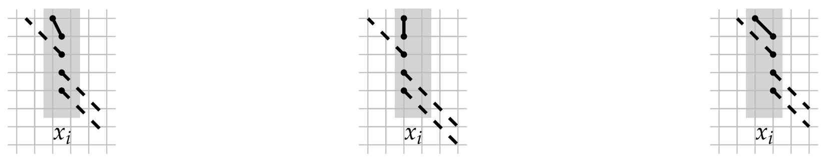
BASIC BUILDING BLOCKS. We will frequently make use of two basic building blocks that rely on the above grid properties.

The first one is a *switch*, i.e., an edge that has exactly two valid embeddings. Let $s = uv$ be an edge within a single grid cell where u is placed on a grid point and v on a non-incident grid edge. We call u the *fixed* and v the *free* vertex of s. Assume that u is in the lower left corner and v on the right edge of the grid cell. Then in any valid d-schematization s has one of only two possible embeddings, either s is embedded horizontally or diagonally. This property justifies the name switch; see Figure 4.3 for an example. *switch*

The second basic concept is *linking* of vertices. We can synchronize two vertices in different and even non-adjacent cells of the grid by assigning them the same x- or y-coordinate. We call two vertices u and v *linked*, if in π either $x_\pi(u) = x_\pi(v)$ or $y_\pi(u) = y_p(v)$. Then the orthogonal order requires that u and v remain linked in any valid embedding. This concept allows us to transmit information on local embedding choices over distances. Two edges $e_i = uu'$ and $e_j = vv'$ are *linked* if there is a vertex of e_i that is linked to a vertex of e_j. We use linking of edges in combination with switches. Namely, we link a switch s via its free vertex with another edge e, as illustrated in Figure 4.3. Then the choice of the embedding of s determines one of the two coordinates of the linked vertices in e. In the case depicted in Figure 4.3 the switch s determines the y-coordinate of both vertices of e; we say that s *pulls* e up (Figure 4.3(b)) or down (Figure 4.3(c)). Such a switch is called a *vertical switch*. Analogously, edges can be pulled to the left and to the right by a *horizontal switch*. *linking*

In the following, if we say that an edge or vertex can only be embedded in a certain way, we always mean that the edge or vertex can only be embedded this way in any valid schematization. Now that we have the switches and the concept of linking available, we will combine them to high-level constructions that model variables and clauses of a PLANAR MONOTONE 3-SAT formula.

VARIABLE GADGETS. The variable gadget for a variable x is a simple structure consisting of a horizontal switch and a number of linked *connector vertices* on consecutive grid lines below the switch, one for each appearance of x or $\neg x$ in a clause of φ. We denote the number of appearances as $t(x)$. All connector vertices share the same x-coordinate in π. Each one will be connected to a clause with a diagonal *connector edge*. A connector edge *connector vertices* *connector edge*



(a) input (b) `true` (c) `false`

Figure 4.4: (a) The input embedding of a variable gadget with three connector vertices, (b) the state `true`, and (c) the state `false`.

spans the same number of grid cells horizontally and vertically and hence can only be embedded at a direction of 135°. Since both vertices of a connector edge are placed on grid lines the connector edge can only have correct direction if the positions of both vertices within their respective grid cells is the same. The upper connector vertices of the variable gadget connect to the gadgets of the negative clauses and the lower connector vertices to the gadgets of the positive clauses. The variable gadget has two states, one in which the switch pulls all vertices to the left (defined as `true`), and one in which it pulls them to the right (defined as `false`). Figure 4.4 shows an example. A variable gadget g_x for a variable x takes up one grid cell in width and $t(x)+1$ grid cells in height, one grid cell for each connector vertex and an additional grid cell for the horizontal switch.

CLAUSE GADGETS. The general idea behind the clause gadget is to create a set of paths whose embedding is influenced by three connector edges. Each of these edges carries the truth value of the connected variable gadget. The gadget consists of three diagonal edges e_2, e_3, e_4, two vertical edges e_1 and e_5 and four switches s_1, s_2, s_3, s_4; see Figure 4.5. For positive clauses, we want that if all its connector edges are pulled to the right, i.e., all literals are `false`, there is no valid embedding of the clause gadget. This is achieved by placing the edges of the gadget in such a way that there are certain points, called *critical points*, where in a valid embedding two different non-adjacent edges can possibly have a vertex. Further, we ensure with the help of switches that each of the three diagonal edges of the clause gadget has exactly one vertex embedded on a critical point in a valid embedding.

critical points

critical edge

Key to the gadget is the *critical edge* e_3 that is embedded with one fixed vertex on a grid point and one loose vertex inside the grid cell A to the top left of the fixed vertex. Edge e_3 is linked with the vertical switch s_2 and the horizontal switch s_4. These switches ensure that the loose vertex must be placed on a free corner of the grid cell A. The three free corners are all critical points. It is clear that there is a valid embedding of e_3 if and only if at least one of the three critical points is available.

We use the remaining edges of the gadget to block one of the critical points of A for each literal that is `false`. The lower vertex of the middle connector edge is placed just to the left of the upper left corner of A such that it occupies a critical point if it is pulled to the right. Both, the left and right connector edges have another linked vertical edge e_1 and e_5 appended. Now, if e_1 is pushed to the right by its connector edge, then the edge e_2 is pushed upward since e_1 and e_2 share a critical point. Due to switch s_1 edge e_2 blocks the lower left critical point of A in that case. Similarly, edge e_4 blocks the upper right critical point of A if the third literal is `false`.

The clause gadget for a negative clause works analogously such that a critical point is blocked for each connector edge that is pulled to the left instead of to the right. It corresponds basically to the positive clause gadget rotated by 180°.

Lemma 4.2 *A clause gadget has a valid embedding if and only if at least one of its literals is* `true`.

Proof. To see this we note that there are five critical points in the gadget and the connector edge of each literal that is `false` blocks one of the critical points. The three edges e_2, e_3, e_4

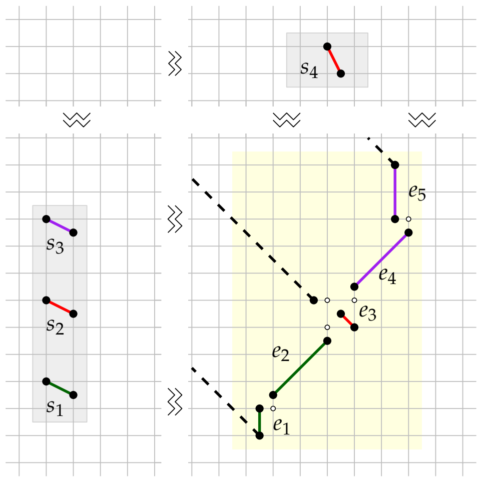


Figure 4.5: Sketch of a clause gadget. Critical points are marked as white disks.

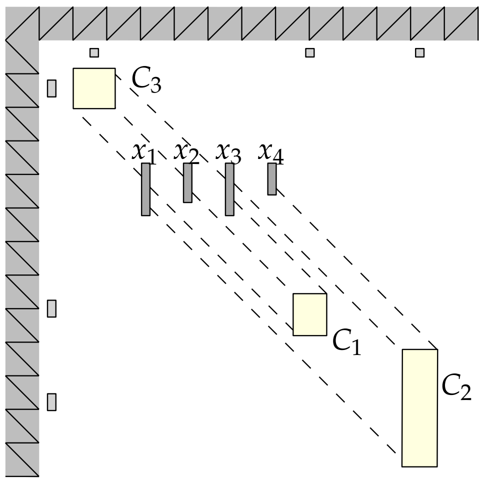


Figure 4.6: Sketch of the gadgets for the instance φ from Figure 4.1.

must also occupy one critical point each. So if all literals are `false`, there are only two critical points remaining for three edges and hence there is no valid embedding of the gadget. Conversely, if there is a valid embedding, then one of the connector edges does not block a critical point and hence the literal that it represents is `true`. □

The size of a clause gadget (not considering the switches) depends on the horizontal distances Δ_1, Δ_2 between the left and middle connector edge and between the middle and right connector edge, respectively. The width of a gadget is 6 and its height is $\Delta_1 + \Delta_2 - 5$.

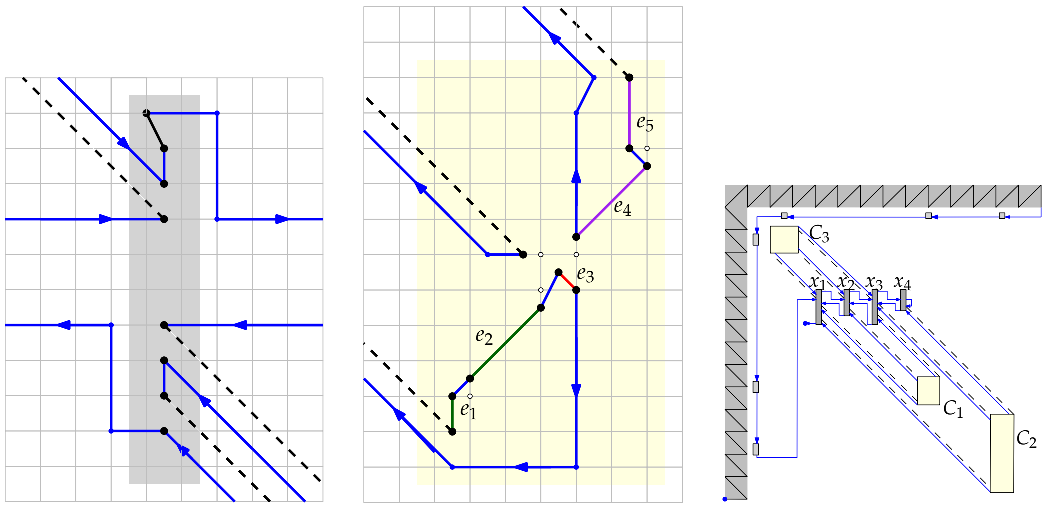
GADGET PLACEMENT. The top and left part of our construction is occupied by the border gadget that induces the grid. The overall shape of the remaining construction is similar to a slanted version of the standard embedding of an instance of PLANAR MONOTONE 3-SAT as in Figure 4.1.

We place the individual variable gadgets horizontally aligned in the center of the drawing. Since variable gadgets do not move vertically, there is no danger of accidentally linking vertices of different variable gadgets by placing them at the same y-coordinates. For the clause gadgets to work as desired, we must make sure that any two connector edges to the same clause gadget have a minimum distance of seven grid cells. This can easily be achieved by spacing the variable gadgets horizontally so that connector edges of adjacent variables cannot come too close to each other.

To avoid linking between different clause gadgets or clause gadgets and variable gadgets, we place each of them in its own x- and y-interval of the grid with the negative clauses to the top left of the variables and the positive clauses to the bottom right. The switches of each clause gadget are placed at the correct positions just next to the border gadget. Figure 4.6 shows a sketch of the full placement of the gadgets for a PLANAR MONOTONE 3-SAT instance φ. Since the size of each gadget is polynomial in the size of the formula φ, so is the whole construction. The above construction finally yields the following theorem.

Theorem 4.1 *The 2-regular Union of Paths Schematization Problem is NP-hard.*

Proof. For any given PLANAR MONOTONE 3-SAT instance φ with variables $X = \{x_1, \dots, x_n\}$ and clauses $C = \{c_1, \dots, c_m\}$ we construct and place the paths for the border, variable and clause gadgets as in Section 4.3.1. From Lemma 4.2 we know that a clause gadget has a valid embedding if and only if it contains a true literal. So the whole construction has a



(a) Variable gadget (b) Clause gadget (c) Sketch of full instance

Figure 4.7: Augmented gadget versions. Additional path edges are highlighted in blue.

valid embedding if and only if every clause gadget has a valid embedding. This is the case if and only if φ is satisfiable.

Both the size of the construction and the time to create it are polynomial in n and m, the number of variables and clauses in φ. □

4.3.2 *Hardness of the Path Schematization Problem*

To prove that 2-regular PSP is also NP-hard we have to show that we can augment the union of paths constructed above by adding additional vertices and edges to form a single simple path that still has the property that it has a valid embedding if and only if the corresponding PLANAR MONOTONE 3-SAT formula φ is satisfiable.

The general idea is to start the path at the lower end of the border gadget and then collect all the switches next to the border gadget. From there we enter the upper parts of the variable gadgets and walk consecutively along all the connector edges into the negative clause gadgets. Once all negative connectors have been traversed, the path continues along the positive connectors and into the positive clauses. The additional edges and vertices must be placed such that they do not interfere with any functional part of the construction.

The only major change is that we double the number of connector vertices in each variable gadget and add a parallel dummy edge for each connector edge. That way we can walk into a clause gadget along one edge and back into the variable gadget along the other edge. We also need a clear separation between the negative and positive connector vertices, i.e., the topmost positive connector vertices of all variable gadgets are assigned the same y-coordinate and we adjust the required spacing of the variable gadgets accordingly. Figure 4.7(a) shows an example of an augmented variable gadget with one positive and two negative connector edges.

The edges e_1–e_3 and e_4, e_5 of each clause gadget are inserted into the path in between the leftmost connector edge and its dummy edge and in between the rightmost connector edge and its dummy edge, respectively. The details are illustrated in Figure 4.7(b). It is clear that by this construction we obtain a single simple path P that contains all the gadgets of our previous reduction. Moreover, the additional edges are embedded such that they do not interfere with the gadgets themselves. Rather they can move along with the flexible parts without occupying grid points that are otherwise used by the gadgets. We conclude:

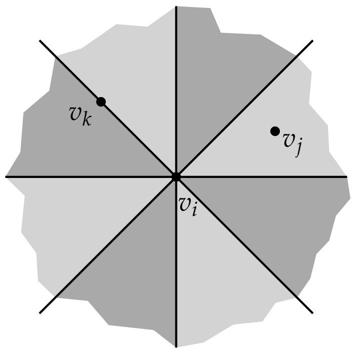


Figure 4.8: Possible relative positions of vertices v_j and v_k with respect to a vertex v_i. The eight sectors are shaded in gray.

Lemma 4.3 *The 2-regular Path Schematization Problem is NP-hard.*

4.3.3 *Membership in NP*

To show that the Path Schematization Problem for $d = 2$ is NP-complete it remains to argue its membership in NP. For this we sketch a non-deterministic polynomial-time algorithm, which guesses a solution that can subsequently be verified in polynomial time. More precisely, we guess a combinatorial structure for an input instance which we then verify in a section step with a linear program (LP).

First, we guess for each pair of vertices their relative position. We distinguish between 16 different relative positions for a vertex pair. More specifically, for a vertex v_i another vertex v_j lies either in one of the eight sectors (shaded in gray) or on one of the eight half lines with origin at v_i (black lines); see Figure 4.8. Further, to help us ensure that the resulting path is not self intersecting, we guess additionally for each pair of path edges which edge is completely contained in one of the half planes induced by the line obtained by extending the other edge. Finally, we also guess on which side of the line the other edge lies. Note that two line segments do not intersect if and only if one segment is fully contained in one of the half planes induced by the other segment's extension to a line.

To verify that the structure guessed as above induces a valid 2-schematization we need to determine whether there exists a schematization that (i) conforms to the structure, (ii) maintains the orthogonal order of the input, (iii) is 2-regular, and (iv) is not self intersecting. Checking whether all path edges have a direction in $\mathcal{C}_2$ can be done straightforwardly as it follows directly from the relative vertex positions. Next we sketch an LP that enables us to check if there exists a valid 2-schematization which conforms to the remaining requirements.

For each vertex v_i of the path we maintain its coordinates in variables x_i and y_i. The first set of constraints for our LP follows directly from the orthogonal order. For all remaining constraints we need to find a way to ensure that a vertex lies to the left/on/to the right of a line. For lines that are horizontal or vertical this is trivial. For the diagonal directions we need to be more careful. To ensure that a vertex v_j lies to the left/on/to the right of a line ℓ through v_i with direction $\alpha \in \mathcal{C}_2$ we make use of the dot product. This yields that $(y_j - y_i) - \tan(\alpha)(x_j - x_i)$ is larger than 0/equal to 0/smaller than 0 if the vertex v_j lies to the left/on/the right of the line ℓ. With this observation we can easily add constraints that ensure that the relative position of each vertex pair is maintained and the path is intersection free.

We do not need an objective function since any feasible solution for our linear program induces a valid 2-schematization of the input. This is sufficient for our purposes. Note that the coefficients in some constraints are $\tan(\alpha)$ which is potentially irrational. However, for

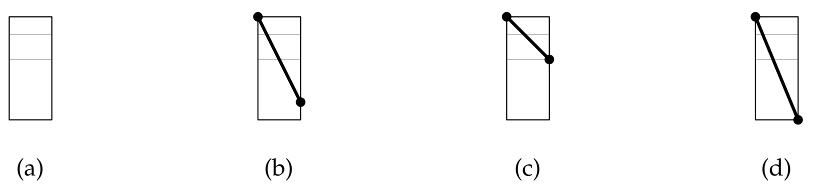


Figure 4.9: (a) A meta cell with its three minor cells for $d = 4$; (b) input embedding of a vertical switch; (c)+(d) the two possible output embeddings of the vertical switch.

the special case $d = 2$ those numbers are rational, and hence, we can use standard linear programming algorithms (e. g, Vaidya's LP algorithm [178]) to check in polynomial time whether the LP admits a feasible solution. Hence, PSP is in NP for $d = 2$.

We can conclude that in combination with Lemma 4.3 the following.

Theorem 4.2 *The 2-regular Path Schematization Problem is NP-complete.*

4.4 NP-COMPLETENESS OF d-REGULAR PSP FOR $d > 2$

In Section 4.3 we established that the 2-regular Path Schematization Problem is NP-complete. Here we show that d-regular PSP is NP-complete for all $d > 2$ by modifying the gadgets of our reduction for $d = 2$.

Theorem 4.3 *The d-regular Path Schematization Problem is NP-complete for any $d > 2$.*

Proof. An obvious problem for adapting the gadgets is that, due to the presence of more than one diagonal direction, the switches do not work any more for uniform grid cells. In a symmetric grid, we cannot ensure that the free vertex of a switch is always on a grid point in any valid d-schematization. This means that we need to devise a different grid in order to make the switches work properly. We construct a new border gadget that induces a grid with cells of uniform width but non-uniform heights. We call the cells of this grid *minor cells* and form groups of $d - 1$ vertically consecutive cells to form *meta cells*. Then all meta cells again have uniform widths and heights. For an example of a meta cell and its minor cells see Figure 4.9(a).

minor cells
meta cells

The $d - 1$ different heights of the minor cells are chosen such that the segments connecting the upper left corner of a meta cell to the lower right corners of its minor cells have exactly the directions that are multiples of $(90/d)^\circ$ and lie strictly between 0° and 90°. This restores the functionality of switches whose fixed vertex is placed on the upper left corner of a meta cell and whose free vertex is on a non-adjacent grid line of the same meta cell. See Figure 4.9(b) for an example of a vertical switch.

BORDER GADGET. The main purpose of the border gadget is to provide a way to control the placement of vertices on discrete positions. In Section 4.3 we used for this a simple gadget, which induced a regular grid. However, since for $d > 2$ there are more than one possible diagonal directions, the grid induced by the simple gadget might be highly irregular. Thus, we need to find a new border gadget that provides a similar functionality as the simple border gadget of Section 4.3 and which can also handle multiple diagonal directions. As for the simple case, the border gadget in this section consists also of a horizontal part B_h and a vertical part B_v. However, the horizontal part B_h of the border gadget consists of two new gadgets, the spiral- and the stairs-gadget, which we describe next.

The spiral gadget consists of a path and its main property is that in any valid d-schematization the first edge of this path must be drawn with the steepest non-vertical direction.

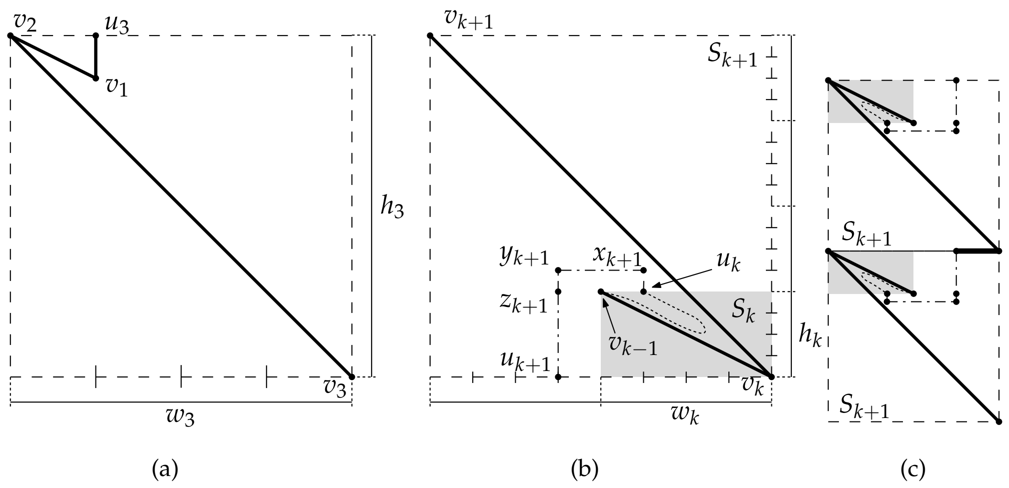


Figure 4.10: The spiral gadget S_k. (a) the spiral gadget S_3 and (b) the construction of S_{k+1} from S_k for $k \geq 3$. The gadget S_{k+1} is rotated by 180° degrees to better highlight the construction. In (c) two spirals in their correct orientation placed on top of each other and connected by the bold edge.

This yields that the width and height of this edge have a fixed ratio. We use this later to ensure that certain vertices have the same distances.

The spiral gadget S_k, where $k \geq 3$, is constructed inductively. All its vertices are contained in the bounding box spanned by the vertices v_{k-1} and v_k. Each gadget S_k has an *entry vertex* v_k and an *exit vertex* u_k which has the same y-coordinate as v_{k-1}. The base case S_3 is depicted in Figure 4.10(a). The gadget S_{k+1} is constructed by adding to S_k the vertex v_{k+1} and the edge $v_k v_{k+1}$ as well as a path $u_k x_{k+1} y_{k+1} z_{k+1} u_{k+1}$; the construction is illustrated in Figure 4.10(b). The vertex x_{k+1} has the same x-coordinate as u_k and is above it. The vertex y_{k+1} has the same y-coordinate as x_{k+1} and is to the left of v_{k-1} and to the right of v_{k+1}. The vertices y_{k+1}, z_{k+1} and u_{k+1} share the same x-coordinate. The vertex y_{k+1} is vertically aligned with x_{k+1}, the vertex z_{k+1} is vertically aligned with u_k, and the vertex u_{k+1} is vertically aligned with v_k. Note that all vertices are contained in the bounding box spanned by v_k and v_{k+1}. Finally, we rotate the result by 180° degrees.

We now discuss the size of a single spiral gadget. We denote the width of a spiral gadget S_k by w_k and its height by h_k. In each step of iterative construction we position the new vertex v_{k+1} such that the edge $v_k v_{k+1}$ spans a box with width $2 \cdot w_k$ and height $4 \cdot h_k$. This directly induces a grid on which we can place the vertices and it implies that we require coordinates in $O(4^k)$ to describe a gadget S_k; see Figure 4.10(b). However, to ensure that we can construct a path that can leave the spiral gadget we need to increase the size of the grid by a factor of 4 both vertically and horizontally. With this increase, vertex coordinates in $O(16^k)$ are sufficient to describe a gadget S_k. Although this might seem problematic, please note that the size of our spiral gadget S_d does only depend on d which is not part of the input and thus the coordinates to describe the spiral gadget are in $O(1)$.

Lemma 4.4 *For any $k \geq 2$ and any valid d-schematization of S_{k+1}, the edge $v_{k+1}v_k$ is steeper than $v_k v_{k-1}$.*

Proof. For $k = 2$ the statement follows since u_3 and v_1 must be embedded above the edge $v_2 v_3$. Now let $k > 2$ and let (S_{k+1}, ρ) be a valid d-schematization. For simplicity we consider the drawing that is obtained by a rotation of 180° degrees. Then the construction looks as depicted in Figure 4.10(b). Observe that this preserves the directions of the edges. Hence, the statement of the lemma holds if v_{k-1} is embedded below the edge $v_k v_{k+1}$. Otherwise, v_{k-1} is embedded above $v_k v_{k+1}$. Since u_k is to the right of v_{k-1}, it follows that u_k is also embedded above $v_k v_{k+1}$. Since u_{k+1} is vertically aligned with v_k, it is embedded below

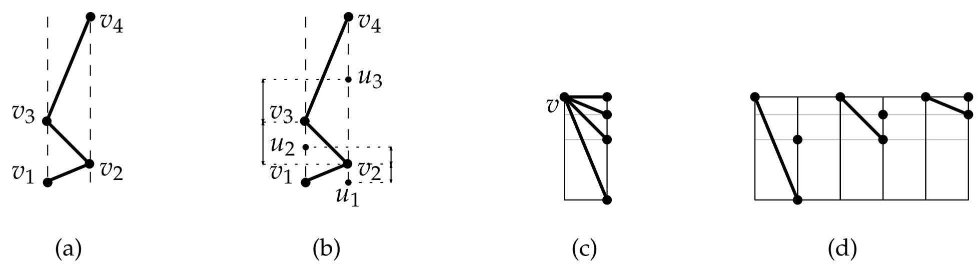


Figure 4.11: Substructures of the border gadget for $d = 4$. (a) The stairs gadget; (b) stairs gadget with additional vertices; (c) non-path vertical border gadget; (d) transformation of (c) into a union of paths.

$v_k v_{k+1}$. The path starting at v_{k-1} and consisting of S_k, u_k, x_{k+1}, y_{k+1}, z_{k+1} and u_{k+1} is completely contained in the bounding box spanned by $v_k v_{k+1}$. On the other hand, this path connects u_k with u_{k+1}, which lie on different sides of $v_k v_{k+1}$. This contradicts the planarity of ρ. □

Lemma 4.5 *For any $d \geq 3$ the spiral gadget S_d has a valid d-schematization. In any valid d-schematization of S_d the edge $v_d v_{d-1}$ has the steepest non-vertical direction.*

Proof. For any $d \geq 3$, the gadget S_3 has a valid d-schematization such that the edge $v_1 v_2$ has direction $90°/d$ and $v_2 v_3$ has direction $2 \cdot 90°/d$. To construct a drawing for S_{k+1} we use a drawing of S_k which uses the directions $90°/d, \dots, (k-1) \cdot 90°/d$. Again, for simplicity we consider the rotated drawing as in Figure 4.10(b). We draw $v_{k+1} v_k$ with direction $k \cdot 90°/d$ such that v_{k+1} is strictly to the left of v_{k-1}. The path from u_k to u_{k+1} is drawn orthogonally. This proves the first statement of the lemma.

We now prove the second statement. Let $d \geq 3$ and let ρ be a valid d-schematization of S_d. Since ρ contains a valid d-schematization of S_k for $k = 3, \dots, d$, it follows from Lemma 4.4 that $v_k v_{k-1}$ is steeper than $v_{k-1} v_{k-2}$ for $k = 3, \dots, d$. Hence, $v_1 v_2, \dots, v_{d-1} v_d$ is a sequence of $d-1$ edges that must be embedded with increasing directions. Since $v_1 v_2$ cannot be embedded horizontally (overlap with u_3) and $v_d v_{d-1}$ cannot be embedded vertically (overlap with u_{k+1}) and since there are only $d-1$ diagonal directions, the claim follows. □

We can place two copies of a spiral vertically above each other such that corresponding vertices have the same x-coordinates and join them by adding a horizontal edge between the last vertex of the lower copy and the first vertex of the upper copy; see Figure 4.10(c). The fact that the direction of the first edge e_1 is fixed shows that this structure is in a sense regular; both spirals have the same width and height.

The spiral provides us with a simple way of creating a construction that contains evenly spaced points. Although, at a first glance, it seems that we might be able to use the spiral gadgets directly to form a grid, this is not the case since the vertices in the interior of the gadget would influence the embedding of the remaining gadgets. Hence, we need another gadget, the *stairs gadget*, which overcomes this drawback, but is not rigid by itself.

A stairs gadget consists of $d-1$ diagonal edges placed in a zig-zag pattern from bottom to top in such a way that per gadget only two x-coordinates are used; see Figures 4.11(a) and 4.11(b). We wish that in any valid schematization the edges $e_i = v_i v_{i+1}$, $i = 1, \dots, d-1$ are all embedded diagonally with increasing directions.

To ensure that the first edge cannot be embedded horizontally, we add an additional vertex u_1 that shares its y-coordinate with v_1 and its x-coordinate with v_2. For each i with $2 \leq i \leq d-1$, we construct two spirals that are joined as described above and such that the starting point of the first spiral has the same y-coordinate as v_i and its endpoint has the same y-coordinate as v_{i+1}. We then place an additional point u_i vertically between v_{i-1} and v_{i+1} that has the same y-coordinate as the endpoint of the upper spiral.

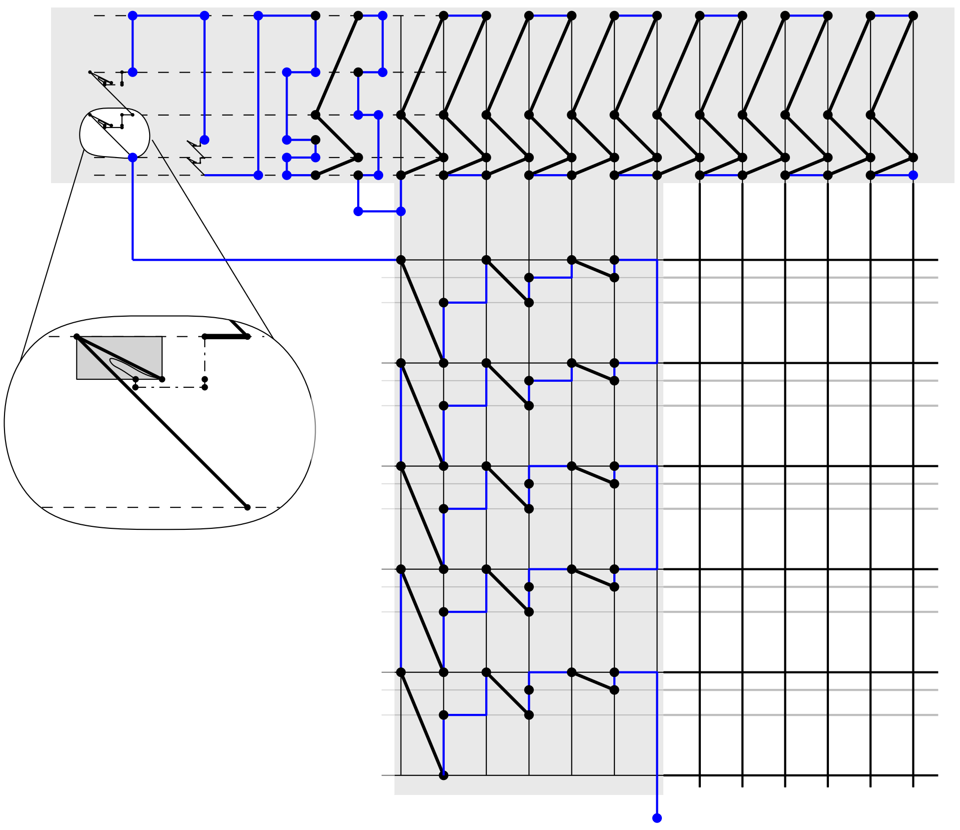


Figure 4.12: Sketch of the border gadget for $d = 4$.

The properties of the spirals imply that the vertical distance between u_i and v_i is the same as the vertical distance between v_{i-1} and v_i in any valid d-schematization. Since v_{i+1} must be embedded strictly above u_i the edge e_{i+1} must be embedded steeper than e_i. As we have only $d-1$ diagonal directions the fact that the stairs gadget has $d-1$ edges shows that each of them is used exactly once and in increasing order along the path.

Placing k identical stairs directly next to each other yields k unit-width grid rows. If we link the corresponding vertices of the stairs gadgets with each other we need to place the vertices u_i only in one of the stairs gadgets. Since we want to combine all gadgets to a path in the end, we use this observation and create a single stairs gadget with the additional vertices to the left of all other stairs gadgets. We can now create a path starting in this leftmost stairs gadget and then connecting the remaining $k-1$ stairs gadgets.

Now that we have established the horizontal part B_h of the border gadget we can assume that all grid columns are of equal width 1. Hence the meta grid cells have height $\tan((d-1)/d \cdot 90°)$. We want to choose the heights of the minor cells such that the line segments from the upper left corner of the meta cell to all grid points on the right side of the meta cell have a direction in $\mathcal{C}_d$; see Figure 4.11(c). This property is necessary for constructing switches as explained later.

Unfortunately, the gadget in Figure 4.11(c) is not a path and hence we cannot use it to induce the required grid structure directly. Instead, we "stretch" the structure and use the concept of linking in order to yield the same result. We use $d-1$ grid columns—one for each diagonal direction—with one "empty" grid column between any two non-empty grid columns; see Figure 4.11(d). To induce ℓ grid rows we repeat this structure ℓ times.

The horizontal and vertical parts of the border gadget are combined by placing the vertical parts such that all vertices are placed on grid lines. A sketch of the complete border gadget for $d = 4$ is shown in Figure 4.12.

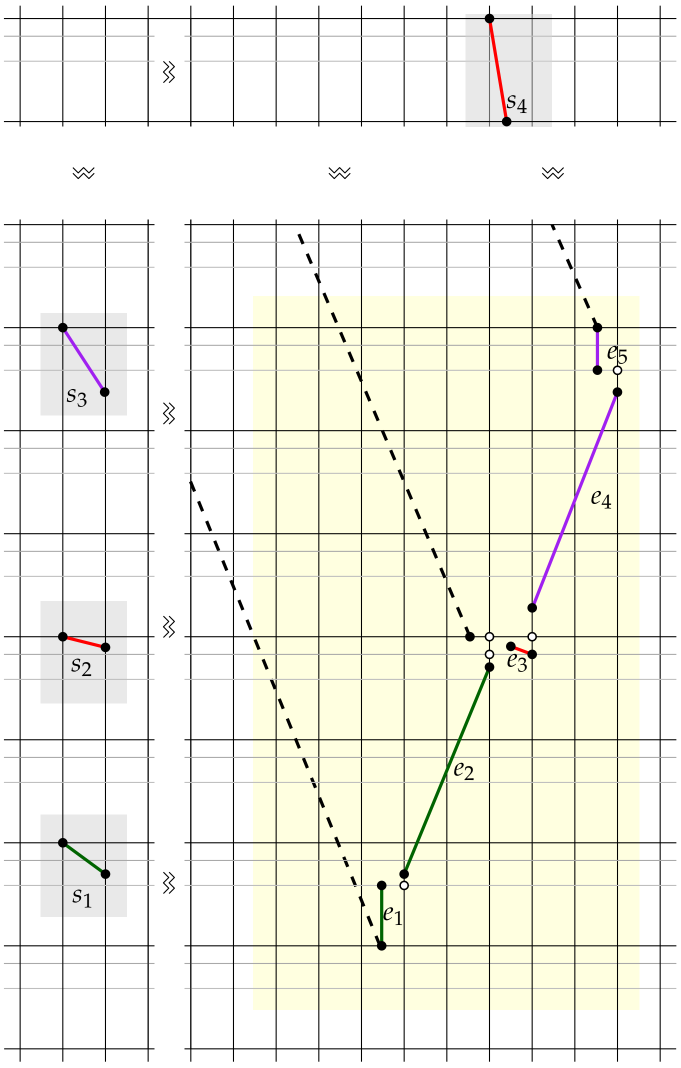


Figure 4.13: Sketch of a clause gadget for $d = 4$.

SWITCHES. Since the border gadget induces a non-regular grid, the switches must be adjusted. Each switch starts at the upper left corner of a meta cell. For a horizontal switch the free vertex is placed on the lower edge of the meta cell. For a vertical switch the free vertex is placed on the right edge of the meta cell in between two horizontal grid lines.

VARIABLE AND CLAUSE GADGETS. The variable gadget is unchanged, given that all its vertices are placed on meta grid lines. The overall structure and functionality of the clause gadgets remains unchanged as well. However, we need to do some slight modifications; see Figure 4.13. The critical edge e_3 is placed in the topmost minor cell of a meta cell. Due to the nature of our border gadget and the induced grid the diagonal edge e_2 starts at the second-to-top minor cell of a meta cell and the diagonal edge e_4 starts at the bottommost minor cell of a meta call. The edge e_1 must have its upper vertex on the lower grid line of the second-to-top minor cell. The edge e_5 must have its lower vertex on the top grid line of the bottommost cell in a meta cell.

GADGET PLACEMENT AND SINGLE PATH. The placement of the gadgets is identical as for $d = 2$. The variable gadgets are placed sufficiently spaced at the same height along the x-axis. The negative clauses are to their upper left and the positive clauses to their lower right. The border gadget is chosen large enough to induce a grid that covers all the gadgets. Combining the gadgets to a single simple path works analogously to the case $d = 2$ and the size of all gadgets is polynomial.

MEMBERSHIP IN NP. It remains to argue that the Path Schematization Problem is in NP for $d > 2$. We use the same idea described in Section 4.3.3. First, we guess the relative vertex positions and the constraints which ensure that the path is not self-intersecting and then formulate the LP. Note that the constraints are essentially the same with the minor difference that coefficients in the constraints can be of the form $\tan(i/(2d) \cdot \pi)$. Those numbers might be irrational which poses a problem for standard LP algorithms. However, since $i/(2d)$ is a rational number and $\tan(r \cdot \pi)$ with $r \in \mathbb{Q}$ is an algebraic number [17, 138] we can use the LP algorithm by Beling [17] to test for feasibility in polynomial time. Hence, we can conclude that the Path Schematization Problem is in NP for $d > 2$. This concludes the proof of Theorem 4.3. □

Corollary 4.1 *The d-regular Path Schematization Problem is NP-complete for any $d \geq 1$.*

Proof. From Theorem 4.2, Theorem 4.3, and the result of Brandes and Pampel [37] for $d = 1$ it follows that the d-regular Path Schematization Problem is
NP-complete for any $d \geq 1$. □

DISCUSSION. We have shown that the d-regular Path Schematization Problem is NP-complete for $d \geq 2$. However, the gadgets in our proofs heavily rely on the concept of linking, which requires vertices to either have the same x- or the same y-coordinates. This is contrast to the proof by Brandes and Pampel [36, 37] which does not rely on this. In fact the authors argue that the proof for $d = 1$ remains valid for points in general position, i. e., no three points lie on the same line. Unfortunately, it seems to be highly non-trivial to adapt our gadgets for this case. It is unclear whether or not it is possible to construct a gadget that is similar to our border gadget in that it produces a regular grid (for $d = 2$). Hence, we cannot make a statement on the complexity of the Path Schematization Problem under the assumption that the input points are in general position. This is an interesting open question which requires additional research.

4.5 MONOTONE PATH SCHEMATIZATION

In the previous sections we considered the problem of deciding for a given embedded path if it admits a d-regular schematization. For the practical application of generating route sketches, we are not only interested if there is such an embedding but we want to determine an embedding with minimum schematization cost, i.e., a d-regular schematization where a maximum number of edges have their preferred direction. Since we know from Corollary 4.1 that the Path Schematization Problem is NP-complete, even if we do not consider schematization cost we cannot hope for an efficient algorithm that determines a d-regular embedding of a given input path, unless P = NP. Hence, in this section we explore the complexity of a more restricted variant of the problem.

We show that, if we restrict the input to an axis-monotone, i.e., paths that are x- or y-monotone, embedded path, the d-regular path schematization problem can be solved efficiently. We devise an algorithm that solves this problem while simultaneously minimizing the schematization cost, and guaranteeing that each edge has length larger than some given minimum length. Moreover, we present a heuristic to extend this approach to general simple paths by first splitting the input path into a minimum number of axis-monotone paths, then applying the algorithm for each path separately, and finally concatenating the paths such that the resulting path is intersection-free.

We begin by stating the exact definition of the problem:

Problem 1 (Monotone-Path Schematization Problem (MPSP)) *Given an embedded and axis-monotone path $(P = (V, E), \pi)$, an integer $d \geq 1$, a minimum length $\ell_{\min}(e)$ for each edge $e \in E$, find a valid d-schematization ρ such that:*

(i) the schematization cost $c(\rho)$ is minimum,

(ii) for every edge $e \in E$ the length $|\rho(e)|$ is at least $\ell_{\min}(e)$, and

As a practical requirement we want to find a path with minimum schematization cost *and* with minimal total path length. Note that schematization cost and total path length are two potentially conflicting optimization criteria. Primarily, we want to find a d-regular schematization that minimizes the schematization cost.

4.5.1 *Solving the Monotone-Path Schematization Problem*

In the following we describe an efficient two-step algorithm that solves MPSP optimally for x-monotone paths. The algorithm can be adapted to handle y-monotone paths straightforwardly. In the first step the algorithm computes a valid d-schematization ρ of the input path (P, π) with minimum schematization cost. In the second step we adjust the embedding ρ that was computed in the first step to ensure the minimum edge length requirement is met and the total path length is minimized.

4.5.1.1 *Minimizing the Schematization Cost*

The goal in the first step of the algorithm is to find a valid d-schematization ρ of (P, π) with minimum schematization cost. To this end we argue that a given special order on the y-coordinates of vertices of (P, π) induces a valid d-schematization of (P, π) with minimum schematization cost among all valid d-schematization of (P, π) that respects this order. We then use this insight to give a dynamic programming algorithm that computes such an order by assigning integer value between 1 and n to the y-coordinate of every vertex of P to generate the output embedding ρ with minimum schematization cost. But first we explain the direction cost considered in this section.

We begin by assigning the preferred direction $\omega_{\mathcal{C}}(e) = \gamma$ to each edge $e \in P$, where $\gamma \in \mathcal{C}_d$ is the direction closest to $\alpha_\pi(e)$. We now group all edges $e = uv$ of P into four categories according to their preferred direction $\omega_{\mathcal{C}}(e)$:

h-edge
1. if $\omega_{\mathcal{C}}(e) = 0°$ and $y_\pi(u) \neq y_\pi(v)$, then e is called horizontal edge (or *h-edge*);

sh-edge
2. if $y_\pi(u) = y_\pi(v)$, then e is called strictly horizontal edge (or *sh-edge*);

v-edge
3. if $\omega_{\mathcal{C}}(e) \neq 0°$ and $x_\pi(u) \neq x_\pi(v)$, then e is called vertical edge (or *v-edge*);

sv-edge
4. if $x_\pi(u) = x_\pi(v)$, then e is called strictly vertical edge (or *sv-edge*).

Using these categories, we define the direction cost for an edge as follows. All edges e with $\omega_{\mathcal{C}}(e) = \alpha_\rho(e)$ are drawn according to their preferred direction and we assign the cost $c_\rho(e) = 0$. For all edges e with $\omega_{\mathcal{C}}(e) \neq \alpha_\rho(e)$ we assign the cost $c_\rho(e) = 1$. An exception are the sh- and sv-edges, which must always be assigned their preferred direction due to the orthogonal ordering constraints. Consequently, we set $c_\rho(e) = \infty$ for any sh- or sv-edge e with $\omega_{\mathcal{C}}(e) \neq \alpha_\rho(e)$.

Note that assigning each edge its preferred direction might result in the following conflict. Consider two subsequent edges e_1, e_2 with $\{\omega_{\mathcal{C}}(e_1), \omega_{\mathcal{C}}(e_2)\} = \{90°, 270°\}$. Assigning such preferred directions would result in an overlap of e_1 and e_2. In this case, we either set $\omega_{\mathcal{C}}(e_1)$ or $\omega_{\mathcal{C}}(e_2)$ to its next best value, depending on which edge is closer to it. This neither changes the solution nor creates new conflicts since in a plane embedding not both edges can have their preferred direction.

In the following we show that for a special kind of order, we call *y-order*, on the y-coordinates of the vertices of (P, π) we can always find a valid d-schematization. Moreover, we show that for a given y-order we can easily construct a valid d-schematization with minimum schematization cost among all valid d-schematizations respecting this order. A y-order for a given path (P, π) is an order on the y-coordinates of the vertices of P that (i) respects the orthogonal order of (P, π) and (ii) for every pair of vertices v, w in P the order tells us if the vertices share the same y-coordinate or if one lies strictly above the other. We are now ready to state the lemma.

y-order

Lemma 4.6 *Let (P, π) be a path with x-monotone embedding, and let $\mathcal{Y}$ be a y-order for (P, π). Then, we can construct a valid d-schematization that respects $\mathcal{Y}$ with minimum schematization cost among all valid d-schematizations of (P, π) that respect $\mathcal{Y}$.*

Proof. Since we assume the input embedding to be x-monotone, the output embedding ρ must be x-monotone, too, as it preserves the orthogonal order of π. So we can assume that $P = (v_1, \ldots, v_n)$ is ordered from left to right in both embeddings. Let ρ' be any orthogonal-order preserving embedding of π. We start with the observation that in ρ' every edge $e = v_i v_{i+1}$ with $\omega_{\mathcal{C}}(e) \neq 0°$ and $y_{\rho'}(v_i) \neq y_{\rho'}(v_{i+1})$ can be embedded with its preferred direction $\alpha_{\rho'}(e) = \omega_{\mathcal{C}}(e)$. This is achieved by horizontally shifting the whole embedding ρ' right of $x_{\rho'}(v_{i+1})$ (including v_{i+1}) to the left or to the right until the direction of e satisfies $\alpha_{\rho'}(e) = \omega_{\mathcal{C}}(e)$. Due to the x-monotonicity of P no other edges are affected by this shift.

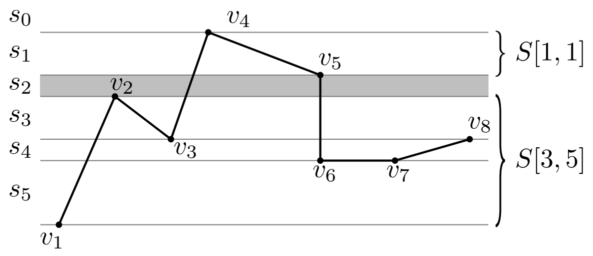
Now, to see how to obtain a valid d-schematization with minimum schematization cost consider the following. Assume ρ is a valid d-schematization of (P, π) that respects $\mathcal{Y}$. For any edge $e = v_i v_{i+1}$ in P that is an h-edge (sh-edge) the direction cost induced by $\mathcal{Y}$ is 1 (∞) if and only if $\mathcal{Y}$ implies $y_\rho(v_i) \neq y_\rho(v_{i+1})$; otherwise the direction cost is 0. Likewise, if e is a v-edge (sv-edge) then the direction cost induced by $\mathcal{Y}$ is 1 (∞) if and only if $\mathcal{Y}$ implies $y_\rho(v_i) = y_\rho(v_{i+1})$; otherwise, by the above shifting argument, we can assign e its preferred direction and hence, its direction cost is 0. Obviously, there can be no valid d-schematization of (P, π) that respects $\mathcal{Y}$ with less schematization cost.

It remains to prove that there exists a valid d-schematization ρ that respects $\mathcal{Y}$. We give a constructive proof. Start by determining y-coordinates according to $\mathcal{Y}$. We assign from bottom to top (according to $\mathcal{Y}$) integer values between 1 and n to the vertices as y-coordinates. Note that two vertices, that according to $\mathcal{Y}$ share the same y-coordinate, get assigned the same y-coordinate. Since we have now assigned each vertex a y-coordinate, and as we have seen, the y-order already determines for which edges we can assign their preferred direction and for which edges this is not possible, it remains to determine the x-coordinates x_ρ of the vertices of P. Note that the x-coordinate of each vertex $v_i \in P$, $0 < i < n$ in ρ depends only on the y-coordinate of its predecessor v_{i-1} and the assigned direction $\alpha_\rho(v_i v_{i+1})$. Hence, we are now able to obtain a valid d-schematization ρ of (P, π) solely based on a given y-order $\mathcal{Y}$. □

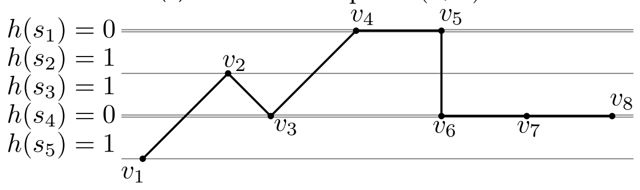
Recall that the aim for the first step of our two-step approach was to determine a valid d-schematization of a given x-monotone embedded input path (P, π). Note that Lemma 4.6 implies we only need to compute a y-order that induces a valid d-schematization with minimum schematization cost to obtain such a schematization ρ.

In the following we give a dynamic programming algorithm that implicitly computes a y-order for an input path (P, π) such that it induces a d-regular schematization of (P, π) with minimum schematization cost. The algorithm assigns each vertex a y-coordinate between 1 and n. The complete valid d-schematization can then be obtained according to the proof of Lemma 4.6.

In order to assign each vertex a y-coordinate between 1 and n, we define $m \leq n - 1$ closed and vertically bounded horizontal strips $s_1, \ldots, s_m$ induced by the set $\{y = y_\pi(v_i) \mid 1 \leq i \leq n\}$ of horizontal lines through the vertices of (P, π). Let these strips be ordered from top to bottom as shown in Figure 4.14(a). Furthermore we define a dummy strip s_0 above s_1 that is unbounded on its upper side. We say that an edge $e = uv$ *crosses* a strip s_i and conversely



(a) Horizontal strips for (P, π)



(b) Strip height assignment and induced output path (P, ρ).

Figure 4.14: Example of (a) an x-monotone embedded input path (P, π) and (b) a 2-regular schematization (P, ρ).

that s_i *affects* e if $\pi(u)$ and $\pi(v)$ lie on opposite sides of s_i. In fact, to determine the cost of an embedding ρ it is enough to know for each strip whether it has a positive height or not since this induces a y-order on the vertices. The algorithm will assign a symbolic height $h(s_i) \in \{0, 1\}$ to each strip s_i, such that the resulting y-order of the vertices induces a valid d-schematization of the input path with minimum schematization cost. Note that sh-edges do not cross any strip but rather coincide with some strip boundary. Hence all sh-edges are automatically drawn horizontally and have no direction costs. We can therefore assume that there are no sh-edges in (P, π).

Let $S[i, j] = \bigcup_{k=i}^{j} s_k$ be the union of the strips $s_i, \ldots, s_j$ and let $\mathcal{I}(i, j)$ be the subinstance of the path schematization problem containing all edges that lie completely within $S[i, j]$. Note that $\mathcal{I}(1, m)$ corresponds to the original instance (P, π), whereas in general $\mathcal{I}(i, j)$ is not necessarily a connected path but may consist of a collection of edges. The following lemma is key to our algorithm.

Lemma 4.7 *Let $\mathcal{I}(i, j)$ be a subinstance of the monotone-path schematization problem and let $s_k \subseteq S[i, j]$ be a strip for some $i \leq k \leq j$. If we assign $h(s_k) = 1$, then $\mathcal{I}(i, j)$ decomposes into the two independent subinstances $\mathcal{I}(i, k-1)$ and $\mathcal{I}(k+1, j)$. The direction costs of all edges affected by s_k are determined by setting $h(s_k) = 1$.*

Proof. We first show that the cost of any edge $e = uv$ that crosses s_k is determined by setting $h(s_k) = 1$. Since u and v lie on opposite sides of s_k we know that $y_\rho(u) \neq y_\rho(v)$. So if e is a v- or sv-edge, it can be drawn with its preferred direction and $c_\rho(e) = 0$ regardless of the height of any other strip crossed by e. Conversely, if e is an h-edge it is impossible to draw e horizontally regardless of the height of any other strip crossed by e and $c_\rho(e) = 1$. Recall that sh-edges do not cross any strips. Assume that $k = 2$ in Figure 4.14(a) and we set $h(s_2) = 1$; then edges v_3v_4 and v_5v_6 cross strip s_2 and none of them can be drawn horizontally.

The remaining edges of $\mathcal{I}(i, j)$ do not cross s_k and are either completely contained in $S[i, k-1]$ or in $S[k+1, j]$. Since the costs of all edges affected by s_k are independent of the heights of the remaining strips in $S[i, j] \setminus \{s_k\}$, we can solve the two subinstances $\mathcal{I}(i, k-1)$ and $\mathcal{I}(k+1, j)$ independently; see Figure 4.14(a). $\square$

OUR ALGORITHM. We are now ready to describe our algorithm for assigning symbolic heights to all strips $s_1, \ldots, s_m$, and thereby determining the integer y-coordinates for the vertices of P, such that the induced embedding ρ has minimum schematization cost. The main idea is to recursively compute an optimal solution for each instance $\mathcal{I}(1,i)$ by finding the best $k \leq i$ such that $h(s_k) = 1$ and $h(s_j) = 0$ for $j = k+1, \ldots, i$. By using dynamic programming we can compute an optimal solution for $\mathcal{I}(1,m) = (P,\pi)$ in $O(n^2)$ time.

Let $C(k,i)$ for $1 \leq k \leq i$ denote the schematization cost of all edges in the instance $\mathcal{I}(1,i)$ that either cross s_k or have both endpoints in $S[k+1,i]$ if we set $h(s_k) = 1$ and $h(s_j) = 0$ for $j = k+1, \ldots, i$. Let $C(0,i)$ denote the schematization cost of all edges in the instance $\mathcal{I}(1,i)$ if $h(s_j) = 0$ for all $j = 1, \ldots, i$. We use an array T of size $m+2$ to store the minimum schematization cost $T[i]$ of the instance $\mathcal{I}(1,i)$. Then $T[i]$ is recursively defined as follows

$$T[i] = \begin{cases} \min_{0 \leq k \leq i}(T[k-1] + C(k,i)) & \text{if } 1 \leq i \leq m \\ 0 & \text{if } i = 0 \text{ or } i = -1. \end{cases} \tag{4.1}$$

Together with $T[i]$ we store the index k that achieves the minimum value in the recursive definition of $T[i]$ as $\kappa[i] = k$. This allows us to compute the actual strip heights using backtracking. Note that $T[m] < \infty$ since, e. g., the solution that assigns height 1 to every strip induces cost 0 for all sv-edges (recall that we can assume that there are no sh-edges in (P,π) and hence assigning height 1 to every strip is a valid solution). Obviously, we need $O(m)$ time to compute each entry in T assuming that the schematization cost $C(k,i)$ is available in $O(1)$ time. This yields a total running time of $O(m^2)$.

The next step is to precompute the schematization cost $C(k,i)$ for any $0 \leq k \leq i \leq m$ such that it can be accessed in $O(1)$ time. Two sources contribute to the schematization cost. The first source is the schematization cost of all edges that are affected by s_k. As observed in Lemma 4.7, all v- and sv-edges crossing s_k have no cost. On the other hand, every h-edge that crosses s_k has cost 1. So we need to count all h-edges in $\mathcal{I}(1,i)$ that cross s_k. The second source is the cost of all edges that are completely contained in $S[k+1,i]$. Since $h(s_{k+1}) = \ldots = h(s_i) = 0$, we observe that any h-edge in $S[k+1,i]$ is drawn horizontally at no cost. In contrast, no v- or sv-edge e in $S[k+1,i]$ attains its preferred direction $\omega_{\mathcal{C}}(e) \neq 0°$. Hence every v-edge in $S[k+1,i]$ has cost 1 and every sv-edge has cost ∞. So we need to check whether there is an sv-edge contained in $S[k+1,i]$ and if this is not the case count all v-edges contained in $S[k+1,i]$.

In order to efficiently compute the values $C(k,i)$ we assign to each strip s_i three sets of edges. Let $H(i)$ (resp. $V(i)$ or $SV(i)$) be the set of all h-edges (resp. v-edges or sv-edges) whose lower endpoint lies on the lower boundary of s_i. We can compute $H(i)$, $V(i)$, and $SV(i)$ in $O(n)$ time for all strips s_i. Then, for $k \leq i$, the number of h-edges in $H(i)$ that cross s_k is denoted by $\sigma_H(k,i)$ and the number of v-edges in $V(i)$ that do *not* cross s_k is denoted by $\sigma_V(k,i)$. Finally, let $\sigma_{SV}(k,i)$ be the number of sv-edges in $SV(i)$ that do *not* cross s_k.

This allows us to compute the values $C(k,i)$, $0 \leq k \leq i \leq m$, recursively as follows

$$C(k,i) = \begin{cases} \infty & \text{if } \sigma_{SV}(k,i) \geq 1 \\ C(k,i-1) + \sigma_H(k,i) + \sigma_V(k,i) & \text{if } k \leq i-1 \\ \sigma_H(k,k) & \text{if } k = i. \end{cases} \tag{4.2}$$

Since each edge appears in exactly one of the sets $H(i)$, $V(i)$, or $SV(i)$ for some i, it is counted towards at most $n-1$ values $\sigma_H(\cdot,i)$, $\sigma_V(\cdot,i)$, or $\sigma_{SV}(\cdot,i)$, respectively. Thus, for computing all the values $C(i,k)$, we need $O(n^2)$ time and require a table size of $O(n^2)$. However, the space requirement can be reduced to $O(n)$ as follows. We compute and store the values $T[i]$ in the order $i = 1, \ldots, n-1$. For computing the entry $T[i]$ we use only the values $C(\cdot,i)$. To compute the next entry $T[i+1]$, we first compute the values $C(\cdot,i+1)$ from $C(\cdot,i)$ and then discard all $C(\cdot,i)$. This reduces the required space to $O(n)$. We can summarize the results in the following lemma.

Lemma 4.8 *There exists an algorithm that computes the array T of schematization costs in $O(n^2)$ time and $O(n)$ space.*

It remains to determine the strip height assignments corresponding to the schematization cost in $T[m]$ and show the optimality of that solution. We initialize all heights $h(s_i) = 0$ for $i = 1, \ldots, m$. Recall that $\kappa[i]$ equals the index k that minimized the value $T[i]$ in (4.1). To find all strips with height 1, we initially set $j = m$. If $\kappa[j] = 0$ we stop; otherwise we assign $h(s_{\kappa[j]}) = 1$, update $j = \kappa[j] - 1$, and continue with the next index $\kappa[j]$ until we hit $\kappa[j] = 0$ for some j encountered in this process. Let ρ be the d-regular schematization of (P, π) induced by this strip height assignment; see Figure 4.14(b). We now show the optimality of ρ in terms of the schematization cost.

Theorem 4.4 *Given an x-monotone embedded path (P, π) and a set $\mathcal{C}$ of d-regular edge directions, our algorithm computes a a valid d-schematization ρ of (P, π) with minimum schematization cost $c(\rho)$.*

Proof. As we have already shown in Lemma 4.6, a given y-order for the vertices of an input path (P, π) induces a valid d-schematization ρ and with minimum schematization cost among all valid d-schematizations respecting the y-order. Since the input path is x-monotone and by construction there are no two adjacent edges with preferred directions $90°$ and $270°$ the embedding ρ is plane. The embedding ρ also preserves the orthogonal order of π since ρ does not alter the x- and y-ordering of the vertices of P; see Figure 4.14(b).

We show that ρ has minimum schematization cost by structural induction. For an instance with a single strip s there are only two possible solutions of which our algorithm chooses the better one. The induction hypothesis is that our algorithm finds an optimal solution for any instance with at most m strips. Consider an instance with $m+1$ strips and let ρ' be any optimal d-regular schematization for this instance. If all strips s in ρ' have height $h(s) = 0$, then by the recurrence relation (4.1) it holds that $c(\rho) = T[m+1] \leq C(0, m+1) = c(\rho')$. Otherwise, let k be the largest index for which $h(s_k) = 1$, in ρ'. When computing $T[m+1]$ our algorithm also considers the case where s_k is the bottommost strip of height 1, which has a cost of $T[k-1] + C(k, m+1)$. If $h(s_k) = 1$ we can split the instance into two independent subinstances to both sides of s_k by Lemma 4.7. The schematization cost $C(k, m+1)$ contains the cost for all edges that cross s_k and this cost is obviously the same as in ρ' since $h(s_k) = 1$ in both embeddings. Furthermore, $C(k, m+1)$ contains the cost of all edges in the subinstance below s_k, for which we have by definition $h(s_{k+1}) = \ldots = h(s_{m+1}) = 0$. Since k is the largest index with $h(s_k) = 1$ in ρ' this is also exactly the same cost that this subinstance has in ρ'. Finally, the independent subinstance above s_k has at most m strips and hence $T[k-1]$ is the minimum cost for this subinstance by induction. It follows that $c(\rho) = T[m+1] \leq T[k-1] + C(k, m+1) \leq c(\rho')$. This concludes the proof. □

ALTERNATIVE COSTS. In this section we have assumed that the cost for an h-edge (a v-edge) is 1 if it is not embedded with its preferred direction. However, our dynamic programming allows us to have different types of cost, even edge specific cost. Recall that the schematization cost for $C(k, i-1)$ depends on $\sigma_H(k, i)$ and $\sigma_V(k, i)$ where $\sigma_H(k, i)$ is the number of h-edges in $H(i)$ that cross s_k, and $\sigma_V(k, i)$ is the number of v-edges in $V(i)$ that do not cross s_k. The sum of the numbers $\sigma_H(k, i)$ and $\sigma_V(k, i)$ is the cost of setting the strip s_k to have height 1 and setting all strips below s_k to have height 0. To enable edge specific cost we can do the following. First, we compute, as before, the sets $H(i)$ and $V(i)$. But when we determine the values for $\sigma_H(\cdot, i)$ and $\sigma_V(\cdot, i)$ we do not simply count the edges in $H(i)$ and $V(i)$, respectively, but we set $\sigma_H(\cdot, i)$ ($\sigma_V(\cdot, i)$) to be the sum of the weights of the edges in $H(i)$ ($V(i)$). Since this is done in a preprocessing step and requires only $O(n)$ time the total time complexity of the algorithm does not change.

In Theorem 4.4 we have shown that our algorithm works for x-monotone paths, and a generalization to axis-monotone paths is trivial. Further, we have argued that we can have edge-specific cost as schematization cost. We conclude this subsection with the following corollary.

Corollary 4.2 *The monotone path schematization problem (Problem 1) for an axis-monotone input path of length n can be solved by an $O(n^2)$-time algorithm that computes a valid d-schematization ρ of minimum schematization cost.*

4.5.1.2 *Minimizing the Path Length*

In the first step of our algorithm we obtained a valid d-schematization ρ of (P, π) with minimum schematization cost in $O(n^2)$ time. However, this does not account for the edge lengths induced by ρ. So in the second step, we adjust ρ such that the total path length is minimized and $|\rho(e)| \geq \ell_{\min}(e)$ for all $e \in P$. We make sure, however, that the orthogonal order and all directions $\alpha_\rho(e)$—and thus also the schematization cost $c(\rho)$—remain unchanged.

Note that we can immediately assign the minimum length $\ell_{\min}(e)$ to every horizontal edge e in the input (P, ρ) by horizontally shifting the subpaths on both sides of e. For any non-horizontal edge $e = uv$ the length $|\rho(e)|$ depends only on the vertical distance $\Delta_y(e) = |y_\rho(u) - y_\rho(v)|$ of its endpoints and the direction $\alpha_\rho(e)$. In fact, $|\rho(e)| = \Delta_y(e)/\sin(\alpha_\rho(e))$. So in order to minimize the path length we need to find y-coordinates for all strip boundaries such that $\sum_{e \in P} |\rho(e)|$ is minimized. These y-coordinates together with the given directions for all edges $e \in P$ (as computed in the previous step) induce the corresponding x-coordinates of all vertices of P.

For each strip s_i $(i = 0, \dots, m)$ let y_i denote the y-coordinate of its lower boundary. For every edge $e \in P$ let $t(e)$ and $b(e)$ denote the index of the top- and bottommost strip, respectively, that is crossed by e. Then $\Delta_y(e) = y_{t(e)-1} - y_{b(e)}$. We propose the following linear program (LP) to minimize the path length of a given $\mathcal{C}$-oriented embedded path (P, ρ).

$$\begin{array}{lrcll}
\text{Minimize} & \multicolumn{4}{l}{\displaystyle\sum_{e \in P,\, \alpha_\rho(e) \neq 0^\circ} \left[\frac{1}{\sin \alpha_\rho(e)} \cdot (y_{t(e)-1} - y_{b(e)}) \right]} \\[2ex]
\text{subject to} & y_{t(e)-1} - y_{b(e)} & \geq & \sin \alpha_\rho(e) \cdot \ell_{\min}(e) & \forall e \in P,\ \alpha_\rho(e) \neq 0^\circ \\
 & y_{i-1} - y_i & > & 0 & \forall s_i \text{ with } h(s_i) = 1 \\
 & y_{i-1} - y_i & = & 0 & \forall s_i \text{ with } h(s_i) = 0
\end{array}$$

We assign to all vertices their corresponding y-coordinates from the solution of the LP. In a left-to-right pass over P we compute the correct x-coordinates of each vertex v_i from the vertical distance to its predecessor vertex v_{i-1} and the direction $\alpha_\rho(v_{i-1}v_i)$. This yields a modified embedding ρ' that satisfies all our requirements: the path length is minimized; the orthogonal order is preserved due to the x-monotonicity of P and the constraints in the LP to maintain the y-order; by construction the directions of all edges are the same in ρ and ρ'; no edge e is shorter than its minimum length $\ell_{\min}(e)$. Hence, ρ' solves Problem 1 and together with Lemma 4.8 and Theorem 4.4 we obtain

Linear programs can be solved efficiently in $O(n^{2.5}L)$ time using a method of Vaidya [178], where n is the length of the path P and L is the number of input bits. Unfortunately, our linear program potentially contains irrational numbers as constant factors and standard LP algorithms cannot efficiently solve such linear programs. One might be tempted to use the linear program algorithm by Beling [17] since all coefficients are either rational or algebraic numbers [138, 176] which at first glance implies a polynomial running time. However, the algorithm has a running time that is polynomial (among other things) in the degree of the algebraic numbers which, in our case, is in $O(d)$. This, however, means that since the space required for representing d is $O(\log d)$ bits, the algorithm requires exponential time in the input size. For practical applications we can simply round those irrational values to an appropriate precision. However, then we cannot make any claims with respect to the optimality of the resulting solution.

4.5.2 *Extension to General Simple Paths*

In the previous section, we showed how to schematize a monotone path. Unfortunately, some routes in road networks are not axis-monotone, however, they can be decomposed into a usually small number of axis-monotone subpaths. So, we propose the following three-step heuristic to schematize general simple paths, which we call the simple path schematization (SPS) algorithm: We first split the input path (P, π) into a minimum number of axis-monotone subpaths (P_i, π_i), where π_i equals π restricted to the subpath P_i. We embed each (P_i, π_i) separately according to Section 4.5.1. Then, we concatenate the subpaths by adding at most three *path-link* edges between the subpaths such that the resulting path (P', ρ) is a simple $\mathcal{C}$-oriented path. Note that this heuristic does not guarantee to preserve the orthogonal order between vertex pairs of different subpaths.

path-link

Splitting an embedded simple path $P = (v_1, \ldots, v_n)$ into the minimal number k of subpaths $P_i, 1 \leq i \leq k$ with the property that each P_i is an axis-monotone path can be done in a straightforward greedy fashion, starting from v_1. We traverse P until we find the last vertices v' and v'' that are not violating the x- and y-monotonicity, respectively. If v' appears later than v'' on P, we set $P_1 = (v_1, \ldots, v')$, otherwise $P_1 = (v_1, \ldots, v'')$. We continue this procedure until we reach the end of P. This algorithm runs in $O(n)$ time and returns the minimal number k of axis-monotone subpaths, indicated by Proposition 1.

Proposition 1 *The greedy path splitting algorithm divides the input path P into a minimal number of axis-monotone subpaths in linear time.*

Proof. Assume $(L_1, \ldots, L_\ell)$ is an optimal solution that divides P into ℓ x- or y-monotone subpaths, and let $(P_1, \ldots, P_k)$ be the solution obtained by the above algorithm. Observe that due to the greedy approach the following two statements hold: (i) the path L_i for the smallest i such that L_i differs from P_i contains fewer vertices than P_i; (ii) there cannot be any path L_m that fully contains some path $P_j = (v_p, \ldots, v_q)$ and also $v_{q+1} \in L_m$. This implies that if there is a path L_m that contains P_j they both share the same last vertex. Combined, we get that there is no sequence of paths $L_1, \ldots, L_i$ containing more vertices than the sequence $P_1, \ldots, P_i$ and hence $k \leq \ell$. □

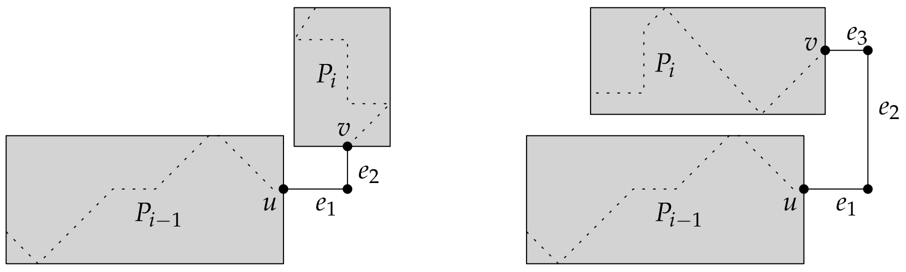
After splitting the input path, we schematize each subpath (P_i, π_i) according to the approach described in Section 4.5.1. We obtain a d-regular schematization ρ_i with minimum schematization cost and minimum path length for each (P_i, π_i). For concatenating these subpaths, we must solve the following problem.

Problem 2 *Given a sequence of k embedded axis-monotone paths (P_i, ρ_i) with $1 \leq i \leq k$, find an embedding ρ of $P' = P_1 \oplus \cdots \oplus P_k$, where $\oplus$ denotes the concatenation of paths, such that*

(i) for each subpath (P_i, ρ_i), the embedding ρ restricted to P_i is a translation of ρ_i and

(ii) (P', ρ) is a simple d-regular path.

Our approach is based on iteratively embedding the subpaths $P_1, \ldots, P_k$. We ensure that in each iteration i the embedding of $P_1 \oplus \cdots \oplus P_i$ remains conflict-free, i. e., it has no self-intersections. We achieve this by adding up to three new *path-link edges* between any two adjacent subpaths P_i and P_{i+1}. For each $1 \leq i \leq k$ let B_i denote the bounding box of (P_i, ρ_i). We show how to construct an embedding ρ of P' such that for any $i \neq j$ we have $B_i \cap B_j = \emptyset$. Consequently, since each individual (P_i, ρ_i) is conflict-free, (P', ρ) is conflict-free as well. A key operation of the algorithm is *shifting* a subpath P_i (or equivalently a bounding box B_i) by an offset $\Delta = (\Delta_x, \Delta_y) \in \mathbb{R}^2$. This is done by defining the lower left corner of each bounding box B_i as its origin o_i and storing the coordinates of P_i relative to o_i, i. e., $\rho(v) = o_i + \rho_i(v)$.



(a) Paths with orthogonal directions (b) Oppositely directed paths

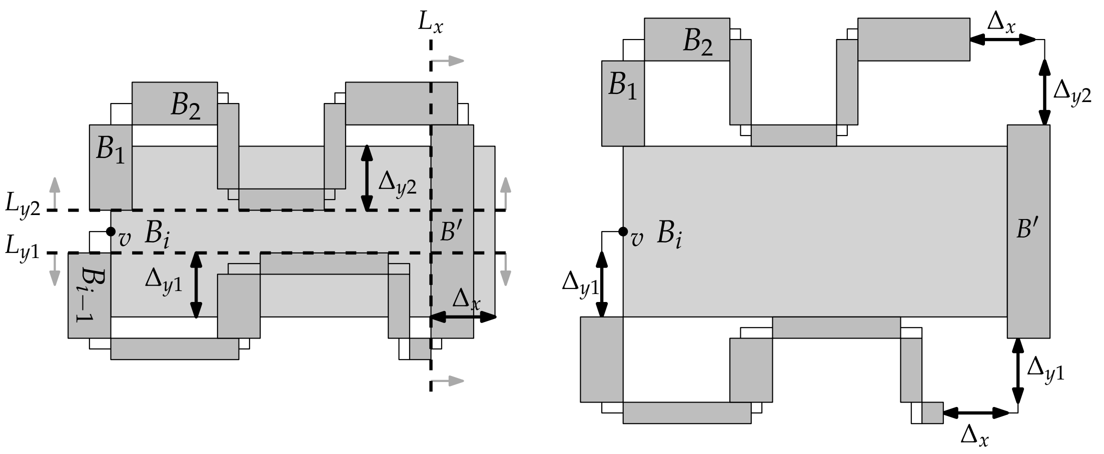
Figure 4.15: Two examples for attaching P_i to P_{i-1} by inserting path-link edges.

Note that shifting preserves all local properties of (P_i, ρ_i), i. e., the orthogonal order as well as edge lengths and orientations.

Each iteration of our algorithm consists of two steps. First, we *attach* the subpath P_i to its predecessor P_{i-1}. To this end, we initially place (P_i, ρ_i) such that the last vertex u of P_{i-1} and the first vertex v of P_i coincide. Then we add either two path-link edges (if the monotonicity directions of P_{i-1} and P_i are orthogonal) or three path-link edges (if P_{i-1} runs in the opposite direction of P_i) between u and v and shift B_i by finding appropriate lengths for the new edges such that $B_{i-1} \cap B_i = \emptyset$. Paths P_{i-1} and P_i are now conflict-free, but there may still exist conflicts between P_i and paths $P_j (j < i - 1)$. These are resolved in a second step that, roughly speaking, "inflates" B_i starting at v until it has reached its original size. Any conflicting bounding boxes are "pushed" away from B_i by stretching some of the path-link edges. In the following, we explain our procedures for attaching a subpath and resolving conflicts in more detail.

ATTACHING A SUBPATH. Without loss of generality, we restrict ourselves to the case that P_{i-1} is an x-monotone path from left to right. Let u be the last vertex of P_{i-1} and v be the first vertex of P_i. If P_i is y-monotone we add a horizontal edge $e_1 = uu'$ with $\alpha_\rho(e_1) = 0°$ connecting u to a new vertex u'. Then we also add a vertical edge $e_2 = u'v$ with $\alpha_\rho(e_1) = 90°$ if P_i is upward directed and $\alpha_\rho(e_1) = 270°$ if it is a downward path. Otherwise, if P_i is x-monotone from right to left, we add two vertices u' and u'' and three path-link edges $e_1 = uu'$, $e_2 = u'u''$, and $e_3 = u''v$ with $\alpha_\rho(e_1) = 0°$, $\alpha_\rho(e_2) = 90°$ if P_i is above P_{i-1} in π or $\alpha_\rho(e_2) = 270°$ otherwise, and $\alpha_\rho(e_3) = 180°$. Note that technically we treat each path-link edge as having its own bounding box with zero width or height. It remains to set the lengths of the path-link edges such that $B_i \cap B_{i-1} = \emptyset$ by computing the vertical and horizontal overlap of B_{i-1} and B_i. Figure 4.15 illustrates both situations. Setting the lengths of the path-link edges leaves us with some degree of freedom. Consider the example depicted in Figure 4.15(a). To resolve the conflict we could set the length of e_2 to 0 and the length of e_1 to the appropriate length. Since in general we want to add as few edges as possible we propose to set the length of the path link edges to 0 whenever possible. Note that this is not always possible. Then, we compute the minimum lengths of e_1 and e_2 that is necessary to resolve the conflict. The edge with the smaller minimum length gets assigned this length, while the other edge length is set to a minimum value. The case depicted in Figure 4.15(b) can be handled similarly.

RESOLVING CONFLICTS. After adding P_i we have $B_{i-1} \cap B_i = \emptyset$. However, there may still exist conflicts with any previously placed box $B_j, 1 \leq j < i - 1$. In order to free up the space required to actually place B_i without overlapping any other bounding box, we push away all conflicting boxes in three steps. For illustration, let P_i be x-monotone from left to right, and let v be the first vertex of P_i.



(a) Before resolving the conflict (b) After resolving the conflict

Figure 4.16: Example for iteratively resolving conflicts induced by attaching P_i. First, we shift everything right of L_x to the right by Δ_x. Then, we shift everything below L_{y1} by Δ_{y1} downward, and finally, we shift everything above L_{y2} upward.

Each bounding box B is defined by its lower left corner $ll(B) = (ll_x(B), ll_y(B))$ and its upper right corner $ur(B) = (ur_x(B), ur_y(B))$. In the first step we identify the leftmost box B' (if any) that is intersected by a line segment that extends from $\rho(v)$ to the right with length equal to the width of B_i. For this box B' we have $ll_y(B') \leq y_\rho(v) \leq ur_y(B')$ and $ll_x(B_i) \leq ll_x(B') \leq ur_x(B_i)$. If there is such a B' let the offset Δ_x be $\Delta_x = ur_x(B_i) - ll_x(B')$.

Now we shift *all* bounding boxes B that lie completely to the right of $ll_x(B')$ to the right by Δ_x. All horizontal path-link edges (which are also considered bounding boxes by themselves) that connect a shifted with a non-shifted path are stretched by Δ_x to keep the two paths connected. Note that there is always a horizontal path-link edge between any two subsequent paths. Since P_i is an x-monotone path from left to right, the bounding Box B_i is currently a horizontal line segment.

Next, we inflate B_i downwards: we first determine the topmost conflicting box B'' (if any) below a horizontal line through $\rho(v)$, i.e., a box B'' whose x-range intersects the x-range of B_i and for which $ll_y(B_i) \leq ur_y(B'') \leq y_\rho(v)$. If we find such a B'' we define the vertical offset $\Delta_{y1} = ur_y(B'') - ll_y(B_i)$. We shift all bounding boxes B that lie completely below $ur_y(B'')$ downwards by Δ_{y1}. All vertical path-link edges that connect a shifted with a non-shifted box are stretched by Δ_{y1} in order to keep the two boxes connected. Again, there is always a vertical path-link edge between any two subsequent paths. Finally, we inflate B_i upwards, which is analogous to the downward inflation. Figure 4.16 shows an example.

The approach explained above first inflates the box to the right, then downwards, and finally upwards. Of course, we could use any strategy of inflating the box. In fact, since different strategies yield different results, we could test a fixed number of strategies and keep the result that minimizes the overall increase in edge lengths.

In the following, we show that our algorithm indeed solves Problem 2.

Lemma 4.9 *Our algorithm computes a conflict-free embedding ρ of $P' = P_1 \oplus \cdots \oplus P_k$.*

Proof. We prove the theorem by induction. By definition, the first path P_1 has a conflict-free embedding. Now, assume that $P_1 \oplus \cdots \oplus P_{i-1}$ is already embedded conflict-free. We attach P_i to P_{i-1}, such that $B_i \cap B_{i-1} = \emptyset$ holds. Thus, P_i and P_{i-1} do not have a conflict. Next, we shift all B_j with $j < i$ such that $B_i \cap B_j = \emptyset$ in the end.

What remains to be shown is that our shifting-operation does not create new conflicts between any two boxes B_j and $B_{j'}$ with $j, j' < i$. Consider, e.g., a shift to the right and assume that after the shift B_j and $B_{j'}$ intersect, while they did not intersect before the shift.

Clearly, if none of the boxes has been moved, they cannot intersect. So either the shift moved both boxes by the same offset Δ_x, or it moved only the rightmost of the two boxes to the right by Δ_x. There is no vertical movement. So in both cases the horizontal distance of the boxes does not decrease and it is impossible that B_j and $B_{j'}$ intersect after the shift. The same observation holds for the vertical shifts. □

Theorem 4.5 *Our algorithm computes a solution (P', ρ) to Problem 2 by adding at most $3(k-1)$ path-link edges to P, where k is the number of given embedded axis-monotone subpath. It can be implemented with a running time of $O(k^2 + n)$.*

Proof. The correctness of the algorithm follows from Lemma 4.9 and due to the fact that the embeddings (P_i, ρ_i) within the bounding boxes B_i remain unchanged. Clearly, we add at most three edges between any two consecutive paths, which shows the bound on the number of path-link edges.

It remains to show the running time of $O(k^2 + n)$. For the position of each bounding box we maintain the coordinates of its lower left and upper right corners. In each iteration, attaching a new subpath requires constant time since it depends only on the overlap of the new bounding box and its immediate predecessor. Resolving all conflicts requires $O(k)$ time per iteration. When adding path P_i we need to check for each box B_j $(j < i-1)$ and the boxes of their respective path-link edges if and how they conflict with B_i. Once the required offset is computed, we shift $O(k)$ boxes by updating their lower left and upper right corners, as well as their origins. This is done in constant time per box and $O(k)$ time in total per iteration. So for all k iterations the running time is $O(k^2)$. Finally, we obtain the embedding ρ of P' by computing the absolute coordinates of all vertices in $O(n)$ time. □

We can now conclude the running time for our simple path schematization algorithm by combining the results from Corollary 4.2 and Theorem 4.5.

Corollary 4.3 *The time complexity of the simple path schematization algorithm is in $O(k^2 + n^2)$.*

Since for practical purposes minimizing the total path length is sensible we additionally need to solve the linear programs described in Section 4.5.1.2. However, for the stated reasons in that section we cannot make any guarantees with respect to the quality of the solution.

Note that if the total number I of conducted shift operations by the algorithm, is sufficiently small, we can do better than $O(k^2 + n^2)$ with the help of more sophisticated data structures. More specifically, if $I < k^2/\log k$ we can use fully dynamic data structures to achieve a time complexity of $O((I + k)\log k + n^2)$ for the part of the algorithm that is independent of the linear program.

As before we store for each bounding box its lower left and upper right coordinates. Additionally, we maintain in total four fully dynamic data structures; two for answering orthogonal line segment intersection queries, and the remaining two for answering orthogonal range reporting queries. One of each two stores the horizontal and the other the vertical segments of all already placed bounding boxes. During the step in which all conflicts are resolved, we use the those data structures to determine only those bounding boxes (and path-link edges) that really intersect a newly placed bounding box. Since this also changes the position of the horizontal and vertical segments of some the already bounding boxes we need to update the data structures accordingly.

We can use data structures proposed by Mortensen [146] to achieve this. Those data structures support insertions and deletions with worst-case time complexity in $O(\log n)$ and queries with worst-case time complexity in $O(\log n + J_i)$, where J_i is the total number of reported segments. In total we need to shift a number $I_i \geq J_i$ of boxes, each of which can be done in $O(\log n)$ time by updating the appropriate data structures. Hence, each iteration requires $O(I_i \log n + \log n)$ time. The total time complexity is $O((I + k)\log k + n^2)$, and the total space requirement for the data structures is $O(n \log n / \log\log n)$.

4.6 A MIXED INTEGER PROGRAM FOR PATH SCHEMATIZATION

From Corollary 4.1 we know that the Path Schematization Problem is NP-complete for every $d \geq 1$. So, unless P = NP, we cannot hope for an efficient exact algorithm to solve the problem. In Section 4.5 we showed that for a restricted class of paths, namely axis-monotone paths, the problem can be solved efficiently. However, the extension of this approach to general paths does not guarantee that the orthogonal order of the whole path is preserved. Thus, first we formulate the problem as a mixed integer linear program (MIP) and then justify our approach through an experimental study by showing that for the application of drawing sketches for real-world driving directions our MIP yields good results in reasonable time frames. The formulation of our MIP is similar to a MIP model for drawing octilinear metro maps [153].

Modeling

Let (P, π) be a given embedded graph, and let d be a positive integer. Recall that for an embedding π of P and an edge e of P we denote by $\alpha_\pi(e)$ the direction of e in π. Besides deciding whether $\mathcal{I}$ admits a valid d-schematization, we transform PSP into the following optimization problem in order to find a valid d-schematization that is as similar to the input path as possible.

Problem 3 (General-Path Schematization Problem) *Given a simple and embedded path $(P = (V, E), \pi)$, an integer $d \geq 1$, and a minimum length $\ell_{\min}(e)$ for each $e \in E$, find a valid d-schematization ρ such that*

(i) for every edge $e \in E$ the deviation of $\alpha_\rho(e)$ from $\alpha_\pi(e)$ is minimal,

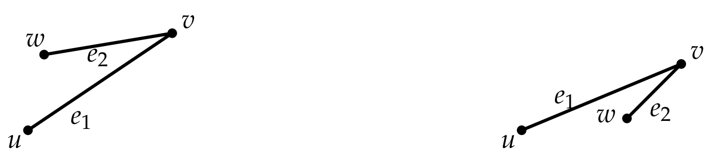
(ii) for every edge $e \in E$ the length $\ell(e)$ is at least a minimum length $\ell_{\min}(e) > 0$, and

(iii) the total length of the path $\sum_{e \in E} \ell(e)$ is minimized.

In the following, we introduce our MIP that solves Problem 3 optimally.

COORDINATE SYSTEM. In order to handle the valid directions of each edge with respect to $\mathcal{C}_d$, we extend the standard Cartesian coordinate system to include one axis for every direction contained in $\mathcal{C}_d$ similarly to [153]. Therefore, we introduce continuous variables $\text{pos}\colon V \times \mathcal{C}_d \to \mathbb{R}$ for every vertex and every possible direction. The value for each coordinate axis $\gamma \in \mathcal{C}_d$ is then defined by $\text{pos}(v, \gamma) = \cos\gamma \cdot x(v) + \sin\gamma \cdot y(v)$. Note, that these are linear constraints, as the values of $\sin\gamma$ and $\cos\gamma$ are constants for fixed d. Moreover, our coordinate system allows us to introduce binary variables $\text{dir}\colon E \times \mathcal{C}_d \to \{0, 1\}$ for deciding whether in the output embedding ρ an edge $e \in E$ is schematized with direction $\gamma \in \mathcal{C}_d$, namely, if and only if $\text{dir}(e, \gamma) = 1$. To enforce that every edge is schematized according to exactly one direction $\gamma \in \mathcal{C}_d$, we add $\sum_{\gamma \in \mathcal{C}_d} \text{dir}(e, \gamma) = 1$ as a constraint for every edge $e \in E$.

ORTHOGONAL ORDER. To maintain the orthogonal order between all pairs of vertices $v_i, v_j \in V$, we can make use of the following observation: Let $v_1 \prec \ldots \prec v_n$ be the input sorted with respect to the x-coordinates of the vertices (with ties broken arbitrarily). Then we introduce a linear number of orthogonal order constraints as follows. We set $\text{pos}(v_i, 0°) \leq \text{pos}(v_{i+1}, 0°)$ if $x_\pi(v_i) < x_\pi(v_{i+1})$ and $\text{pos}(v_i, 0°) = \text{pos}(v_{i+1}, 0°)$ if $x_\pi(v_i) = x_\pi(v_{i+1})$. Analogously, the same constraints are independently introduced for the y-coordinates of the vertices and the values $\text{pos}(v, 90°)$ for all $v \in V$.



(a) Input path. (b) Valid d-schematization for $d = 4$.

Figure 4.17: Example where edge order is violated

EDGE DIRECTIONS AND LENGTHS. To ensure that every edge $e \in E$ is embedded according to the direction γ for which $\text{dir}(e, \gamma) = 1$ holds, we make use of our extended coordinate system in the following way. The edge $e = uv$ is embedded according to $\gamma \in \mathcal{C}_d$ if and only if u and v have the same coordinate on both orthogonal axes $\gamma + 90°$ and $\gamma + 270°$. Thus, for every $e = uv \in E$ and every $\gamma \in \mathcal{C}_d$ we add the constraints $\text{pos}(v, \gamma + 90°) - \text{pos}(u, \gamma + 90°) \leq M(1 - \text{dir}(e, \gamma))$ and $\text{pos}(v, \gamma + 270°) - \text{pos}(u, \gamma + 270°) \leq M(1 - \text{dir}(e, \gamma))$. Here, M is a sufficiently large constant, which we choose to be the maximum possible distance in our coordinate space. This has the effect that the constraints are trivially satisfied for $\text{dir}(e, \gamma) = 0$ and that they enforce the correct direction for $\text{dir}(e, \gamma) = 1$. Additionally, we set the minimum length for every edge by adding constraints $\text{pos}(v, \gamma) - \text{pos}(u, \gamma) \geq -M(1 - \text{dir}(e, \gamma)) + \ell_{\min}(e)$. Again, if M is sufficiently large, these constraints are relevant only if $\text{dir}(e, \gamma) = 1$. In order to minimize the total path length, we define variables $\text{len}\colon E \to \mathbb{R}_0^+$ as an upper bound on the length of each edge $e = uv \in E$ by introducing for every $\gamma \in \mathcal{C}_d$ a constraint $\text{pos}(v, \gamma) - \text{pos}(u, \gamma) \leq \text{len}(e)$. Note that minimizing $\text{len}(e)$ through the objective function will result in tight upper bounds on the actual edge lengths, i. e., $\text{len}(e) = \ell(e)$.

To minimize the deviation of the schematized direction of every edge $e \in E$ to its input direction, we add continuous variables $\text{dev}\colon E \to \mathbb{R}_0^+$ that represent the deviation cost of e according to its embedding in ρ. We set $\text{dev}(e) = \sum_{\gamma \in \mathcal{C}_d} c(e, \gamma)\,\text{dir}(e, \gamma)$, where $c\colon E \times \mathcal{C}_d \to \mathbb{R}_0^+$ is some user-defined deviation cost function. In our implementation we set $c(e, \gamma)$ to the distance in terms of the number of directions in $\mathcal{C}_d$ between the selected direction and the direction in $\mathcal{C}_d$ that is closest to the input direction $\alpha_\pi(e)$.

EDGE ORDER. Although the orthogonal order usually maintains the user's mental map appropriately, there can be cases where the orthogonal order alone is not sufficient. Consider the example displayed in Figure 4.17. The path in Figure 4.17(b) is a valid d-schematization of the input path for $d = 4$, but the order of the edges at vertex v has changed. For route sketches this would mean that instead of a left turn, the schematized path depicts a right turn. This is clearly undesirable. To avoid this problem we iterate over all inner-path vertices and add constraints where necessary. For simplicity we disregard in the following the orientation of the path's edge and assume that each edge incident to a vertex is oriented away from the vertex. First, observe that an edge e incident to a vertex v must lie, in every valid d-schematization, in the same quadrant as in the input with respect to v. Hence, to avoid the discussed problem it is sufficient to consider only those vertices where its incident edges lie in the same quadrant. Now, consider an inner-path vertex v with incident edges e_1 and e_2. Without loss of generality we assume that both edges lie in the first quadrant (to v's upper right) and e_1 has, in the input, the larger direction. Then, for each of smallest the $d - 2$ diagonal directions γ in the first quadrant we add the constraint $\text{dir}(e_2, \gamma) \leq \sum_{90° \geq \gamma' > \gamma} \text{dir}(e_1, \gamma')$. Finally, since e_2 must not be assigned the largest diagonal direction we also require that $\text{dir}(e_2, 90°/d \cdot (d-1)) = 0$.

Further, to avoid overlapping of edges that are incident to the same vertex, we use a similar approach. Note that for edges e_1 and e_2 incident to the same vertex v to overlap both edges are either in the same quadrant with respect to v or in adjacent quadrants. For simplicity we assume again that for each vertex that we consider, both incident edges are oriented away from it. If both edges lie in the same quadrant with respect to v, then the constraints above already ensure that no overlap can occur. If the edges are in adjacent quadrants with respect to v, then we only require $\text{dir}(e_1, \gamma) + \text{dir}(e_2, \gamma) = 0$ for the direction $\gamma \in \mathcal{C}_d$ that is contained in both quadrants.

PLANARITY. In the output embedding ρ, edges may be stretched and embedded with directions different from the input embedding. Thus, we have to ensure that the output embedding is still crossing-free. Hence, we introduce two types of planarity constraints. First, for two adjacent edges $e_i = uv$ and $e_j = vw$ we observe that e_i and e_j only intersect, if $\gamma(e_i) = \gamma(e_j) + 180°$. Thus, for every pair of adjacent edges e_i, e_j and every $\gamma \in \mathcal{C}_d$ we add a constraint $\text{dir}(e_i, \gamma) + \text{dir}(e_j, \gamma + 180°) \leq 1$.

For the remaining non-adjacent pairs of edges $e_i = u_i v_i, e_j = u_j v_j$ we make use of the following observation. The edges e_i and e_j do not intersect if there is at least one coordinate axis $\gamma \in \mathcal{C}_d$ according to which both u_j and v_j lie beyond u_i and v_i. Thus, we introduce binary variables $\text{sep}\colon E \times E \times \mathcal{C}_d \to \{0, 1\}$ selecting for each pair of non-adjacent edges the axis γ for which the previous observation needs to be true. We can now enforce planarity by adding four constraints for every pair e_i, e_j and every $\gamma \in \mathcal{C}_d$. We set $\text{pos}(X, \gamma) - \text{pos}(Y, \gamma) \geq -M(1 - \text{sep}(e_i, e_j, \gamma)) + \delta$, where $X \in \{u_i, v_i\}$, $Y \in \{u_j, v_j\}$, and $\delta \in \mathbb{R}_0^+$ is a user-defined minimum distance to be kept between non-adjacent edge-pairs.

Note that this approach to enforce planarity requires $O(|E|^2)$ binary variables and constraints. However, in our experiments on real-world driving directions, we observe that most of the planarity constraints seem to be unnecessary in the sense that the optimal solution without planarity constraints already yields a crossing-free path. Thus, we may add planarity constraints in a lazy fashion by an iterative process: We start by computing a solution without planarity constraints. Then, we test whether the solution has any intersections and add planarity constraints for these specific conflicting edge pairs. We then recompute a solution with the new set of constraints until we finally obtain a crossing-free embedding ρ.

OBJECTIVE FUNCTION. Subject to all the constraints described above we minimize the following objective function:

$$\text{Minimize} \qquad \sum_{e \in E} \text{dev}(e) + \frac{1}{M} \sum_{e \in E} \text{len}(e).$$

Note that the coefficient $1/M$ ensures that our primary goal is to minimize the total angular deviation as modeled by $\sum_{e \in E} \text{dev}(e)$, while minimizing the total path length becomes the secondary goal.

4.7 EXPERIMENTS

In this section we give a brief experimental evaluation of both approaches we discussed in Sections 4.5 and 4.6. For the MIP approach we generated 1,000 shortest paths (with respect to the travel time metric) in the German road network by selecting source and target vertices uniformly at random. Although our approach requires planar input graphs and a quickest path in a road network is not necessarily planar, we can simply planarize the input path by adding for each intersection of the path a vertex at the intersection point. However, our heuristic approach described in Section 4.5.2 can only handle simple input paths. Hence, we generate a second data set that contains quickest paths without intersections. For the second data set we again select source and target vertex in the German road network uniformly at random, discard a path if it is self-intersecting, and stop when we have 1,000 quickest paths without self-intersections.

The average number of vertices of a path is between 995.2 and 998.3 vertices for both data sets, hence, a schematization would yield a route sketch with far too much detail. Therefore, in a preprocessing step, we simplify each path using the Douglas-Peucker algorithm [72] but keep all important vertices such as points of road category changes. Moreover, we shortcut self-intersecting subpaths whose lengths are below a threshold. We set this threshold to 200m in our experiments. For the SPS algorithm we used the linear program described in Section 4.5.1.2 to reduce the total path length of each axis-monotone path individually.

This is to remove over- or underpasses near slip roads of highways, which are considered irrelevant for route sketches or would be depicted as an icon rather than a loop. Figure 4.18 illustrates an example of a route sketch obtained by our MIP and as comparison a route sketch generated by the heuristic described in Section 4.5.2. It clearly illustrates how a schematic route sketch, unlike the geographic map, is able to depict both high-detail and low-detail parts of the route in a single picture.

We performed our experiments on a single core of an AMD Opteron 2218 processor running Linux 2.6.34.10. The machine is clocked at 2.6 GHz, has 16 GiB of RAM and 2×1 MiB of L2 cache. Our implementation is written in C++ and was compiled with GCC 4.5.0 using optimization level 3. As MIP solver we use Gurobi 5.0.2.We performed our experiments on a single core of an AMD Opteron 2218 processor running Linux 2.6.34.10. The machine is clocked at 2.6 GHz, has 16 GiB of RAM and 2×1 MiB of L2 cache. Our implementation is written in C++ and was compiled with GCC 4.5.0 using optimization level 3. As MIP solver we use Gurobi 5.0.2. We have published the source code as a proof of concept implementation at `http://i11www.iti.kit.edu/_media/projects/routesketch.zip` under the GPL 3.0 License[2].

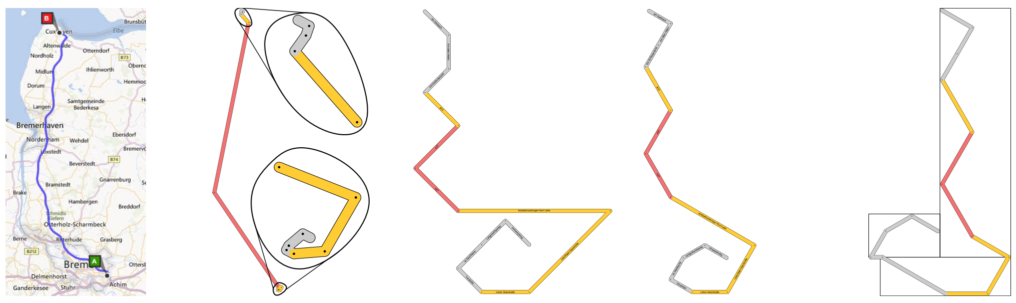
4.7.1 *Case Study*

Comparing Figures 4.18(c) and 4.18(d) indicates that using the parameter $d = 2$ for the schematization process yields route sketches with a very high level of abstraction while using $d = 3$ results in route sketches which approximates the overall shape of the route much better. For example, the route sketch in Figure 4.18(c) seems to suggest that there is a $90°$ turn while driving on the highway but this is not the case. The route sketch depicted in Figure 4.18(c) resembles the original route much better. Generally, we found that using the parameter $d = 3$ yields route sketches with a high degree of readability. See Figure 4.18(d) for an example.

For comparison, we show the output computed by the heuristic method described in Section 4.5.2 in Figure 4.18(e). Recall that the heuristic subdivides a simple input path into maximal axis-monotone subpaths that are subsequently merged into a single route sketch. In order to avoid intersections of the bounding boxes of the monotone subpaths, additional edges of appropriate length are inserted into the path; the orthogonal order is preserved only within the monotone subpaths. This may lead to undesired effects in sketches of non-monotone routes.

FURTHER EXAMPLES. In Figures 4.19 and 4.20 are two example routes as displayed in Bing Maps (a), simplified route determined by the Douglas-Peucker algorithm (b), schematized by our MIP (c), and schematized by our SPS algorithm (d). Figure 4.19(d) shows an example of where the SPS algorithm resolved the intersection between the bounding boxes of both axis-monotone paths by adding a long horizontal path link edge. In Figure 4.20 we observe little difference between the optimal solution computed by our MIP and the solution obtained by the SPS algorithm. In this case no path link edge is necessary. Note that in this example the path computed by the SPS algorithm is shorter

2 http://www.gnu.org/licenses/gpl-3.0



(a) (b) (c) (d) (e)

Figure 4.18: A sample route from Bremen to Cuxhaven in Germany: (a) output of Bing Maps, (b) simplified route after preprocessing, using $\varepsilon = 2^{-4}$ (with manually highlighted start and destination), (c) output of our MIP for $d = 2$ (time to compute solution: 193 ms), (d) output of our MIP for $d = 3$ (time to compute solution: 126 ms), and (e) output of the heuristic from Section 4.5 (time to compute solution: 5.6 ms). Different colors depict different road categories. Note that in contrast to our solution, details at the beginning and the end of the route cannot be seen in the geographically accurate drawings.

than the path computed by the MIP. This is due to the orthogonal order that here requires the grey horizontal line segment to have the same (or a larger) y-coordinate than the vertex connecting the two yellow line segments in the destination area of Konstanz.

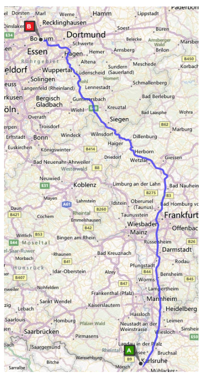
4.7.2 *Experimental Evaluation*

In this section we evaluate hard facts about two suggested approaches. We begin with the mixed integer linear program.

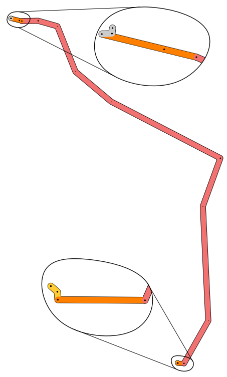
mixed integer linear program. In the first experiment we examine the performance of our MIP for $d = 3$ subject to the amount of detail of the route, as controlled by the distance threshold ε between input and output path in the path simplification step. We also set the threshold for removing self-intersecting subpaths to ε. Tables 4.1(a) and 4.1(c) reports our experimental results for values of ε between 2^{-1} and 2^{-5}.

For the first data set with decreasing ε, the lengths of the paths increase from 19.4 vertices to 30.6 vertices on average. This correlates with the running time of our MIP which is between 141.77 ms and 373.96 ms for the instances. We observe that a value of $\varepsilon = 2^{-3}$ is a good compromise between computation time and amount of detail in practice. Further, adding planarity constraints in a lazy fashion pays off since we need less than 1.14 iterations on average. Thus, most of the paths admit a planar schematization without any planarity constraints at all. The number of infeasible instances, i. e., paths without a valid 3-schematization, is 0.4 %. Finally, there is very little difference between the results for the first data set reported in Table 4.1(a) and for the second data set reported in Table 4.1(c).

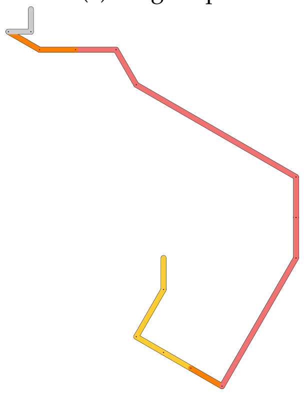
The second experiment evaluates the performance of our MIP when using different values of d, see Table 4.1(b) and Table 4.1(d). We fix $\varepsilon = 2^{-3}$. While for rectilinear drawings with $d = 1$ we need 32.62 ms to compute a solution, the running time increases to 723.84 ms when using $d = 5$. Interestingly, more than half of the paths do not have a valid rectilinear schematization. By allowing one additional diagonal direction ($d = 2$), the number of infeasible instances significantly decreases to roughly 1.5 %. Adding further diagonal directions, i. e., increasing d to a value larger than 2, does not seem to have a significant impact on the percentage of infeasible instances. We observe that in terms of infeasible



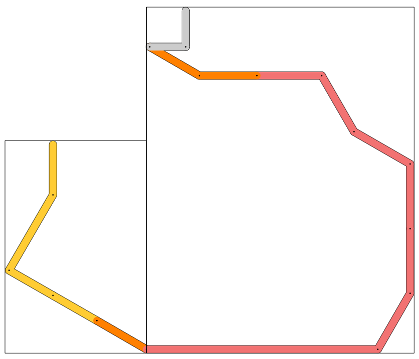
(a) Bing Maps



(b) Simplified input

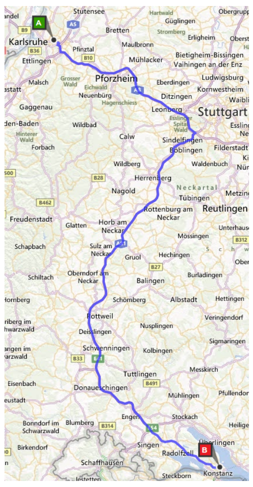


(c) MIP

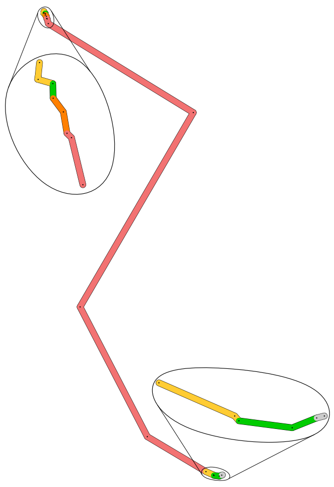


(d) SPS algorithm

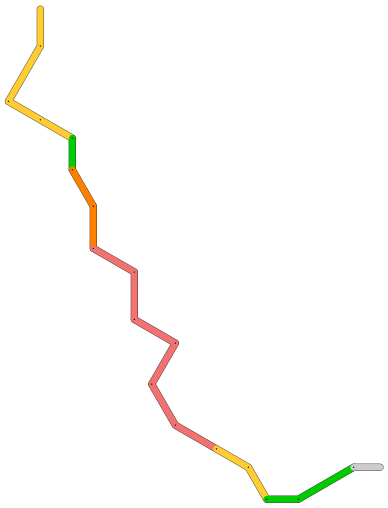
Figure 4.19: Route from Bochum to Karlsruhe. Parameters used: $\varepsilon = 2^{-4}$, and $d = 3$. Time to compute solutions with MIP and SPS algorithm are 262ms and 5.8ms, respectively.



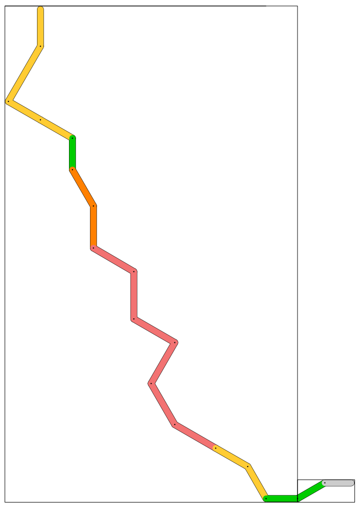
(a) Bing Maps



(b) Simplified input



(c) MIP



(d) SPS algorithm

Figure 4.20: Route from Karlsruhe to Konstanz. Parameters used: $\varepsilon = 2^{-3}$ and $d = 3$. Time to compute MIP solution: 262ms. Time to compute with heuristic: 5.23ms.

Table 4.1: Results obtained by running 1,000 random queries in the German road network and schematizing the simplified paths with our MIP. The tables reports average path lengths (number of vertices), percentage of infeasible instances, average number of iterations, and average running times. The Tables (a) and (b) shows the result for the data sets where self-intersecting input paths are planarized, while Tables (c) and (d) shows the result for the data sets where input paths are non self-intersecting.

ε	~length	% inf	~iter	~time [ms]
2^{-1}	19.4	0.40	1.06	141.77
2^{-2}	19.8	0.40	1.06	156.14
2^{-3}	20.8	0.40	1.07	165.76
2^{-4}	23.4	0.40	1.06	206.60
2^{-5}	30.6	0.40	1.06	373.96

(a) Varying ε with $d = 3$ (intersections).

d	% inf	~iter	~time [ms]
1	56.20	1.21	32.62
2	1.50	1.11	125.56
3	0.40	1.07	165.76
4	0.40	1.05	291.47
5	0.40	1.05	723.84

(b) Varying d with $\varepsilon = 2^{-3}$ (intersections). The average path length after simplifying with is 20.8 vertices.

ε	~length	% inf	~iter	~time [ms]
2^{-1}	19.3	0.00	1.05	132.29
2^{-2}	19.7	0.00	1.05	142.73
2^{-3}	20.7	0.00	1.05	155.69
2^{-4}	23.2	0.00	1.05	186.04
2^{-5}	30.4	0.00	1.05	358.77

(c) Varying ε with $d = 3$ (no intersections).

d	% inf	~iter	~time [ms]
1	52.40	1.24	35.49
2	1.00	1.09	121.04
3	0.00	1.05	155.69
4	0.00	1.04	176.15
5	0.00	1.04	188.05

(d) Varying d with $\varepsilon = 2^{-3}$ (no intersections). The average path length after simplifying with is 20.7 vertices.

instances and average number of iterations there is little difference between the two tested data sets. However, the running time for $d = 5$ for the first data set is 723.84 ms while it is only 188.05 ms for the second data set. This is most likely caused by more difficult instances that are not included in the second data sets because of self-intersections.

It seems surprising that for $d = 1$ there roughly half of all paths in the evaluation do not have a valid rectilinear schematization. This is due to the very limited schematization options coupled with the constraints posed by maintaining the orthogonal order. There are many examples of even very short paths which do not have a valid rectilinear schematization. One such example is depicted in Figure 4.21. If the edge e_1 is assigned direction 90°, then all vertices must share the same x-coordinate, but w must be below v which forces e_1 and e_2 to overlap. Otherwise, if e_1 is assigned 0°, then all vertices must share the same y-coordinate. By similar argumentation as before, the edges e_1 and e_2 must overlap yielding a non-valid rectilinear schematization. Adding only one diagonal direction to the set $\mathcal{C}$ resolves this problem. Hence, it seems that the problem of finding a valid schematization becomes significantly easier when diagonal directions are permitted.

SPS ALGORITHM. Similar to the first experiment, in the third experiment we examine the performance of our heuristic approach described in Section 4.5.2, the simple path schematization (SPS) algorithm. Again, we set $d = 3$ and vary the value of ε. Note that

Table 4.2: Results obtained by running 1,000 simple paths obtained by random quickest path queries in the German road network and schematizing the simplified paths with our SPS algorithm. The table reports the average number of axis-monotone paths per quickest path, the average number of added path link edges to the path, the average part of the total path length that is contributed by the path link edges, average percentage of vertex pairs for which the orthogonal order is maintained, and average running time. We consider varying values for ε with $d = 3$.

ε	∼mon. path.	∼add. edges	∼add. edge length[%]	∼orth. order[%]	∼time [ms]
2^{-1}	3.35	0.55	8.1	92.51	5.16
2^{-2}	3.40	0.56	8.1	92.48	5.22
2^{-3}	3.46	0.57	7.6	93.12	5.89
2^{-4}	3.57	0.57	7.5	93.94	5.87
2^{-5}	4.01	0.60	6.0	95.69	6.57

contrary to the MIP, which always computes an optimal solution (if there exists one), for our heuristical approach there are no infeasible instances (given that the input paths are simple). However, there are two possible problems by the output generated with the algorithm. First, there is no guarantee that the orthogonal order of the input path is preserved. Second, during the process we may need to add several edges to the path. Hence, we report additionally to average running time of the algorithm two parameters: average percentage of vertex-pairs for which the orthogonal order is violated and the average number of edges that is added to the path. Table 4.2 reports our experimental results for values of ε between 2^{-1} and 2^{-5}.

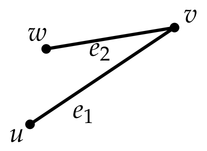
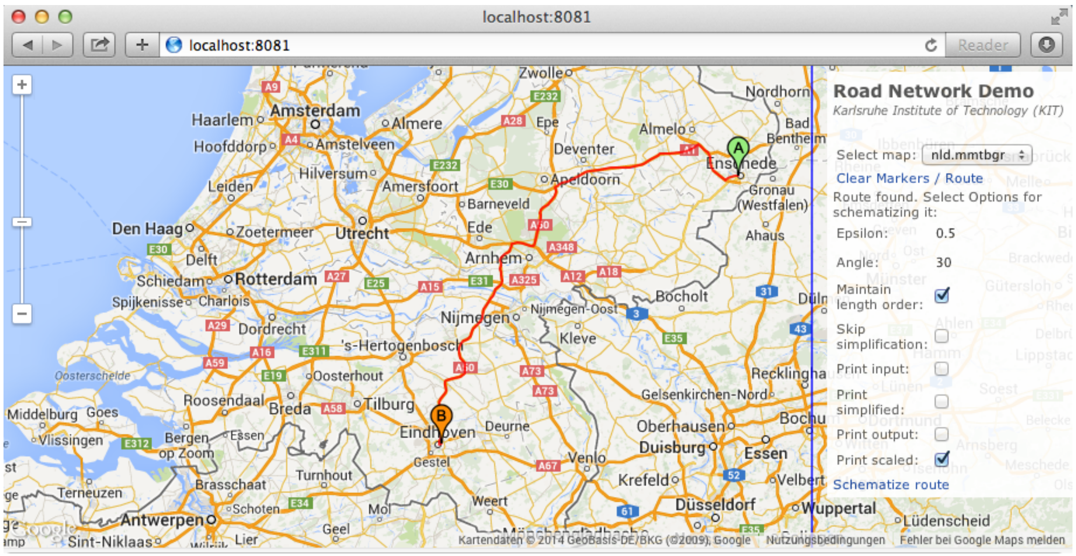


Figure 4.21: Path that has no valid rectilinear schematization.

We observe that with decreasing values of ε there is a slight increase in the average number of monotone paths per quickest paths from 3.35 to 4.01. This correlates with a slight increase in the average number of added path link edges by our algorithm, which increases from 0.55 to 0.60. Note that this number may seem surprisingly low since our algorithm adds at least two edges to connect two axis-monotone paths. However, in our experimental analysis we observed that often the bounding boxes of the axis-monotone paths do not overlap, which means the path link edges have length 0 and thus we can omit them completely. On the other hand, we observe that if path link edges are added that they contribute a non-negligible portion to the total path length (between 6% and 8.1%). Further, we note that the number of vertex pairs for which the orthogonal order is preserved also increases slightly with decreasing values of ε from 92.51% to 95.69%. This is most likely due to the increased number of vertices in the input path which decreases the effect of a single vertex that violates the orthogonal order of the input. Finally, we observe that the running time also increases slightly with decreasing values of ε from 5.16 ms to 6.57 ms which is caused by the increased number of vertices in the input.

We have seen that with our MIP formulation, which guarantees an optimal result, we can obtain good schematizations in a reasonable time frame. Further, the number of infeasible instances (i.e., instances for which there exists no solution) is relatively small. If a fast execution time is more important than optimality, the SPS algorithm presents a viable alternative that produces good results, and in our experiments had an average running time between 5 ms and 7 ms, while the MIP required for the same routes between 132.29 ms and


localhost:8081
Reader
A
B
Enschede
Eindhoven
Kartendaten © 2014 GeoBasis-DE/BKG (©2009), Google
Nutzungsbedingungen
Fehler bei Google Maps melden

Road Network Demo
Karlsruhe Institute of Technology (KIT)

Select map: nld.mmtbgr
Clear Markers / Route
Route found. Select Options for schematizing it:

Epsilon:	0.5
Angle:	30
Maintain length order:	✓
Skip simplification:	☐
Print input:	☐
Print simplified:	☐
Print output:	☐
Print scaled:	✓

Schematize route

Figure 4.22: Web interface for our algorithms. Background provided by Google Maps. The route from "A" to "B" is from Enschede to Eindhoven.

358.77 ms. However, the SPS algorithm can only be used for simple input paths. Another downside of the SPS algorithm is the proportion of the total path length that is contributed by the path link edges. To avoid this it may be a good alternative to use an approach to resolve conflicts that is not solely based on the bounding boxes of the axis-monotone paths.

4.7.3 *Web Interface*

While we conducted all experiments with a command line interface, this is for our application not user-friendly at all. This is why we implemented (mainly done by student helper Philipp Loewner) a web interface, i. e., a web server that provides a simple interface to interact with the map and the algorithms. An example is depicted in Figure 4.22, where a route from Enschede (NL) to Eindhoven (NL) is shown. The background tiles (i. e., the map) are provided by Google Maps, while the computation of the shortest path, as well as the remaining interface is provided by our program. The user can simply set a route by interacting with the map as usual (dragging and zooming are supported operations by Google Maps), and then use a left-click to determine start and destination on the map. The web server then immediately computes a shortest path by using Dijkstra's algorithm [61], and displays it on the map. In the right column there are several options for the user to modify the resulting schematization. The value of *epsilon* influences the degree of simplification (this is the ε used for the Douglas-Peucker algorithm), and the *angle* is the smallest non-zero angle that is permissible as edge direction, i. e., the value of *angle* is $90°/d$. For the web interface we also have an option called *maintain length order*. When activated it ensures that no edge in the final output embedding can be longer than another edge if it was not already longer in the input embedding. This might lead to more visually appealing results. When clicking *Schematize route* the MIP is started and the output is generated (the web interface can also produce the output generated by the SPS algorithm).

4.8 CONCLUSION

Motivated by drawing route sketches in road networks, we studied the problem of d-regular schematization of embedded paths (PSP). In our problem definition we aimed for two main goals: In order to preserve the user's mental map we used the concept of orthogonal order, and to reduce the visual complexity we restricted the valid edge directions to integer multiples of $(90/d)^\circ$. We analyzed the complexity of the problem and showed that PSP is NP-complete for $d \geq 2$. In combination with a previous result of Brandes and Pampel [36, 37] it follows that PSP is NP-complete for every integer $d \geq 1$. Unfortunately, our proof does not work anymore when the input points are in general position. This is in contrast to the proof by Brandes and Pampel. Hence, the complexity of the Path Schematization Problem for $d > 1$ when all input points are in general position is open, but in general the problem is NP-complete.

Nonetheless, we presented two approaches to generate route sketches. The first one is a polynomial-time algorithm that computes for axis-monotone paths and arbitrary integer d a valid d-schematization with minimum schematization cost, for any d. We used this algorithm to derive a heuristic to generate d-schematizations for arbitrary simple paths. An experimental evaluation showed that this approach can very quickly generate outputs with only limited violations of the orthogonal order. However, it is restricted to simple input paths. Our second practical approach to generate route sketches uses mixed integer linear programming (MIP). To generate readable drawings, we minimized the total path length in the objective function of our MIP while ensuring a certain minimum length for each individual edge. Finally, we conducted an experimental evaluation on real-world data in the German road network for both approaches. They revealed that we are indeed able to compute visually appealing solutions of the PSP within approximately 1 sec for reasonable values of $d \leq 5$.

Although maintaining the orthogonal order of the vertices is a common way to preserve the mental map of the user, the orthogonal order has some downsides. It makes it difficult to find valid schematizations (it is NP-complete to decide whether or not a path has a schematization that maintains the orthogonal order) and sometimes the orthogonal order prohibits schematizations which appear to be sensible. To overcome this problem it might make sense to investigate relaxed variants of the orthogonal order, in which a vertex only affects vertices that are "close" to it. Thus, the orthogonal order is locally maintained, which might be sufficient to preserve a global mental map. Since we use the orthogonal order only for maintaining the mental map, an interesting direction is to study alternative ways to model mental map preservation.

5

COLUMN-BASED GRAPH LAYOUTS

We consider orthogonal upward drawings of directed acyclic graphs with nodes of uniform width but node-specific height. One way to draw such graphs is to use a layering technique as provided by the Sugiyama framework [175]. To overcome one of the drawbacks of the Sugiyama Framework, namely, unnecessary edge crossings caused by an unfortunate layer assignment of the nodes, Chimani et al. integrated their layer-free upward crossing minimization algorithm [55] into the Sugiyama framework [56]. However, one drawback of the Sugiyama framework still remains. If the heights of the nodes are non-uniform, the result of the approach can be a non-compact layout. In contrast, we avoid both of these drawbacks by integrating layer-free upward crossing minimization into the topology-shape-metrics (TSM) framework introduced by Tamassia [177]. Our approach, in combination with an algorithm by Biedl and Kant [28] lets us generate *column-based layouts*, i. e., layouts where the plane is divided into uniform-width columns and every node is assigned to a column.

We study the complexity of the individual steps of the layout process systematically and propose efficient algorithms with provable guarantees. We show that our column-based approach allows generating visually appealing, compact layouts with few edge crossing and at most four bends per edge. Furthermore, the resulting layouts exhibit a high degree of symmetry and implicitly support edge bundling. We justify our approach by an experimental evaluation based on real-world examples.

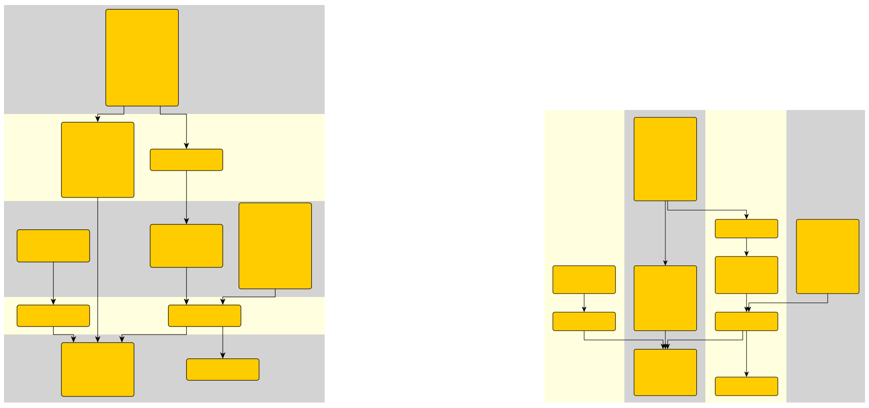
This chapter is based on joint work with Gregor Betz, Christof Mathies, Ignaz Rutter, and Dorothea Wagner. Preliminary results have been published in [27, 26, 71].

5.1 INTRODUCTION

One of the most well-known approaches for drawing directed acyclic graphs (DAGs) is the Sugiyama framework [175]. It consists of three steps, each of which can be solved individually: (i) layer assignment, (ii) determining relative positions within each layer to reduce edge crossings and (iii) positioning the vertices and edges. In the first step, the nodes are assigned to layers such that the target of each edge is in a layer above its source. The second step reduces the number of edge crossings by changing the order of the nodes within each layer. Finally, coordinates are assigned to the nodes as well as to bend-points of the edges.

The Sugiyama framework suffers mainly from two drawbacks. First, computing an unfortunate layering of the nodes in step (i) can enforce many crossings in step (ii) which would not be necessary if another layering was chosen. Second, when applied for drawing graphs where the nodes are depicted as two-dimensional shapes (e. g., rectangles), a single node with large height can lead to non-compact layouts due to the way layers are defined; see Figure 5.1(a) for an example. One approach that resolves the first problem is the layer-free upward crossing minimization approach by Chimani et al. [55]. However, its integration into the Sugiyama framework [56] leaves the second problem.

The motivation for the problems investigated in this paper stems from the application Argunet [166] that is used to create, display, and edit so called *argument maps*. Argument



(a) A layer-based layout (computed by yEd). (b) A column-based layout (our approach).

Figure 5.1: A layer-based layout and a column-based layout of the same graph.

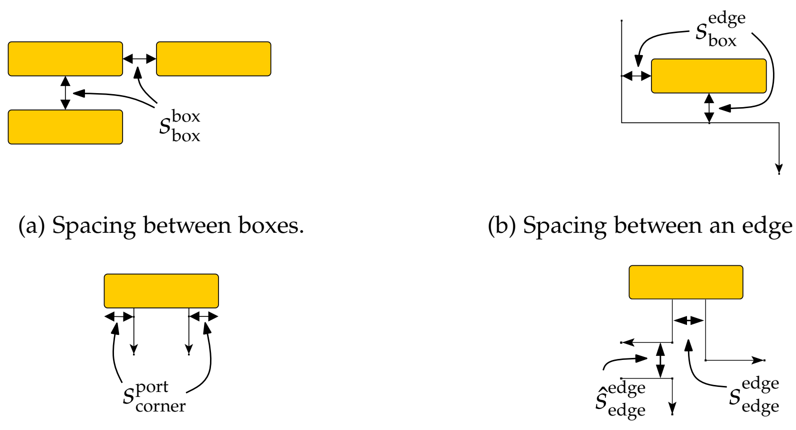
maps are used to visualize the structure of debates or argumentational analyses [48], and usually depicted as directed graphs with nodes of uniform width but node-specific height. Thus, we focus on orthogonal upward drawings of DAGs whose nodes are represented as rectangular boxes of uniform width. Graph drawing with given node sizes has been explored by Di Battista et al. [65]; they consider planar graphs with a fixed embedding and arbitrary node sizes. We deal with non-planar graphs without a given topology but with the restriction that the nodes have uniform width. For minimizing the number of crossings we use the algorithm by Chimani et al. [55] but, instead of integrating it into the Sugiyama framework [56], we integrate it into *topology-shape-metrics framework (TSM)* invented by Tamassia [177]. The TSM framework can be seen as a three-phase-method for orthogonal graph drawing: (i) fixing a planar embedding, (ii) computing an orthogonal description and (iii) compaction of the layout and coordinate assignment.

The integration of layer-free upward crossing minimization into the TSM framework lets us construct *column-based graph layouts*. In a column-based layout the plane is divided into disjoint columns of uniform width that corresponds to the uniform width of the boxes. The boxes and edges are then assigned to these columns. Thereby, vertical edge segments always run within a column, whereas horizontal edge segments may span several columns. Due to the columns the final layouts have a clear look, which is exemplified in Figure 5.1(b). The same graph with a layer-based layout computed by the graph editor yEd is shown in Figure 5.1(a).

column-based graph layouts

However, our approach differs from the usual TSM based approach in one important detail. While we compute in the first step the topology, we relax it already in the second step and consider only the left-to-right order of each node's outgoing edges and disregard crossing dummy nodes. Because of the relaxation of the topology, the final layout can have a higher number of crossings than the topology computed in the first step. However, this allows us to better optimize the vertical edge length in order to obtain more compact layouts. Thus, by relaxing the topology, we increase the significance of minimizing the number of bends and the total edge length as opposed to minimizing the number of edge crossings. This is reasonable since it has been established that crossings where edges cross at large angles (in this case *right angle crossings*) are only a minor obstacle for humans to understand the relationship between nodes in graphs [119].

We consider the following drawing style for our graph layouts: The drawings we generate are orthogonal drawings of DAGs whose nodes are represented by boxes of uniform width and individually prescribed heights. We require that edges leave their sources at the bottom and enter their targets at the top, and we further require that each edge is a monotonic


$s_{\text{box}}^{\text{box}}$
$s_{\text{box}}^{\text{edge}}$
$s_{\text{corner}}^{\text{port}}$
$\hat{s}_{\text{edge}}^{\text{edge}}$
$s_{\text{edge}}^{\text{edge}}$

(a) Spacing between boxes.

(b) Spacing between an edge and a box.

(c) Spacing between the ports of a box and its corners.

(d) Spacing between two edges.

Figure 5.2: Illustration of the five types of spacing constraints.

ports

minimum spacing constraints

downward polyline (we call the points where the edges enter/leave the boxes *ports*). We refer to such a drawing as an upward drawing. Note that upward drawings usually requires strict monotonicity, we however, lessen this restriction in this chapter to simple monotonicity, thus allowing also horizontal segments on the edges. We also include *minimum spacing constraints* that ensure that there is a minimum space between two boxes, a box and an edge, and between ports and corners of boxes, respectively. We denote the values of these space constraints by $s_{\text{box}}^{\text{box}}$, $s_{\text{box}}^{\text{edge}}$, and $s_{\text{corner}}^{\text{port}}$, respectively. We allow to specify two different spacing constraints for the case of two parallel edge segments: one for the distance between two edge segments whose edges have a common source or target and one for the remaining cases. We denote the former by $s_{\text{edge}}^{\text{edge}}$ and the latter by $\hat{s}_{\text{edge}}^{\text{edge}}$. By using these two different minimum spacings we enable *edge bundling*. For an illustration of the different types of minimum spacing constraints see Figure 5.2.

Besides the mentioned drawing style, our approach guarantees certain properties that are aimed at creating visually appealing drawings. When looking at a single box of the layout, it is important that its predecessors can be found quickly. In order to give them a unifying look, we want as many predecessors as possible of each box to be *vertically aligned* at their bottom, which is similar to the requirement for an approach for layered graph drawing by Brandes and Köpf [35]. Additionally, we show how to assign nodes into columns such that, for each node v, its assigned column is the median of the columns assigned to the incoming edges of v as well as the median of its outgoing edges. We call this property *local symmetry*. Local symmetry makes it easy to identify a node's predecessors and successors, and provides well-structured looking layouts. Finally, we guarantee that each edge of the resulting layout will have at most *four bends*.

While satisfying these properties, we optimize the following three criteria, which are frequently employed as optimization goals in graph drawing [66]: (i) minimize the number of edge crossings, (ii) minimize the number of bends and (iii) minimize total edge length.

A different approach to generate drawings of graphs that can be used to produce hierarchical or layered layouts is *constrained graph layout* [114], which is a general approach for drawing graphs. An input to constrained graph layout consists of the graph, a set of constraints for the coordinates of the vertices, and as third parameter suggested values for the variables corresponding to the coordinates of the vertices [75]. Constraints on the coordinates of the vertices can be used to model requirements similar to those we have for the drawings in this paper (e. g., minimum spacing constraints, columns). There have been several practical applications based on this general approach to layout different kind of graphs, e. g., dynamic biological networks [168], or engineering diagrams [74].

We note that, although our motivation stems from generating layouts for argument maps, our approach appears to be applicable to other kinds of diagrams such as the UML class diagrams. UML class diagrams depict the internal structure of a software system and are used extensively in software engineering [32]. A UML class diagrams has for each class a node, which is depicted as a box with height that usually depends on the number variables and methods of the class. Each class node may have an individual prescribed width, but it is not unreasonable to set a uniform value for the width of all boxes (which we require for our approach to work).

There have been several approaches to automatically layout UML diagrams already. However, they are mainly an adaption of layering approaches [169, 79], and thus may suffer from the same problems all such layered drawings may suffer. However, Eiglsperger et al. [80] propose two algorithms, both based on the TSM framework, to draw UML class diagrams with orthogonal edges. They generate a mixed-upward planarization of the input graph. For their algorithm they do not need to restrict the width of the boxes to some uniform value which is necessary for our approach to work.

We note that our approach does not directly support the drawing of UML class diagrams as there additional requirements compared to argument maps, (e. g., labeling of edges, hyperedges). Including these requirements might be an interesting problem for further research.

CONTRIBUTION AND OUTLINE. Our main result is an algorithmic approach to generate column-based graph layouts for directed acyclic graphs. This approach is based on the topology-shape-metric framework and utilizes Chimani et al.'s layer-free upward crossing minimization. Contrary to the standard approach of the TSM framework, we do not fix the topology computed in the first step but use it only as a basis for computing a column-assignment of the nodes in the second step. In the last step we only use this column assignment and ignore the topology from the first step. This lets us focus on optimizing other criteria than the number of edge crossings.

Note that we use the "upward" terminology throughout this chapter, because it is established in the graph drawing community. Nevertheless, since this drawing style originates from a particular application—which requires that all edges are directed downwards—all figures in this work have downward directed edges.

The remainder of the chapter is structured as follows. In the next section we give a short introduction to the TSM framework and our approach. Sections 5.3–5.5 are structured along the TSM framework. In these sections we present our complexity results and give a detailed description of the algorithmic approaches we use in each step of the framework. Finally, in Section 5.6, we present a brief experimental evaluation of our approach.

5.2 PRELIMINARIES

The *TSM framework* consists of the three basic steps *topology*, *shape*, and *metrics*. In the first step an embedding of the input graph is computed with the goal of minimizing the number of crossings in the final layout. For non-planar graphs, crossings are replaced by dummy nodes of degree 4, which we refer to as *crossing dummies*. The planarization together with its embedding is called *topology*. Generally, the topology is fixed throughout the remaining steps of the algorithm. In the second step, the *shape* of the final layout is optimized with respect to the fixed topology. The shape is the assignment of bends to edges (and the angles of these bends). Usually, the goal of this step is to minimize the number of bends. *TSM*

Finally, in the step *metrics*, the edge lengths, the node dimensions, and the node positions are determined.

Our approach to compute column-based graph layouts is based on the TSM framework. This lets us split the problem of generating a graph layout into smaller subproblems and either solve them optimally or apply heuristics if they are NP-hard. In the topology step, we minimize the number of crossings while enforcing that all edges are directed upwards. The result of this step is a so-called upward planar representation, which prescribes the left-to-right order of the edges incident to each node separately for in- and outgoing edges. During the shape computation we build upon this upward planar representation in order to compute a column assignment of the nodes and edges, i. e., we divide the plane into disjoint columns of uniform width and assign each node and each edge to a column. The column assignment induces four different properties—namely local symmetry, orthogonal edges, at most four bends per edge, and the port distribution. In the last step of the topology-shape-metrics-framework we only rely on the column assignment computed in the second step to compute the final coordinates of the boxes and edge bends. Our approach here differs to the common usage of the topology-shape-metrics-framework in that we *do not* fix the topology computed in the first step. We found out that during the shape and metrics phase, minor changes to the topology can improve the aesthetics of the final layout. This lets us assign higher significance to bend and total edge length minimization than in usual applications of this framework.

The main result of this chapter is stated in the following theorem.

Theorem 5.1 *For a given connected directed acyclic graph $G = (V, A)$ a locally-symmetric, column-based layout with at most four bends per edge can be computed in $\mathcal{O}(|A|^3|V|)$ time and $\mathcal{O}(|A||V|^2)$ space.*

In the next sections we study the complexity of each step and give detailed descriptions our algorithmic approaches. Combining the main theorems of each section proves Theorem 5.1.

5.3 TOPOLOGY

In this section we deal with the first step of the TSM framework, which aims to find a topology of the input graph with few crossings. Before we come to the main part of the topology-step, we conduct a preliminary step in which the input graph $G = (V, A)$ (which is directed and acyclic) is transformed into an s-t-graph $\widehat{G} = (\widehat{V}, \widehat{A})$. An *s-t-graph* is a directed graph having a single source and a single sink. The transformation is done by adding a super source $\hat{s}$ and a super sink $\hat{t}$ to G and connecting them with all of its original sources and sinks, respectively. In the end, when converting the layout of $\widehat{G}$ to G, we simply omit $\hat{s}$ and $\hat{t}$ as well as all their incident edges. Note that for the algorithm by Chimani et al. [55], which our approach is based on, it is sufficient to consider s-T-graphs (which in contrast to s-t-graphs may have several sinks). The reason why we consider s-t-graphs instead is due to the special requirement of layouting argument maps that there are no nodes placed above sources and below sinks. We refer to this as the *free sources/sinks* property. For other applications, this requirement can be ignored, possibly resulting in more compact layouts.

s-t-graph

free sources/sinks

While planarity of arbitrary graphs can be tested efficiently, testing upward planarity is NP-complete for general graphs [94]. Therefore, upward crossing minimization is NP-complete as well. However, for s-t-graphs, the test for upward planarity can be done in polynomial time [24], and thus upward crossing minimization might be efficiently solvable for s-t-graphs. We refer to this problem as s-t-Upward Crossing Minimization. However, there is an easy reduction from Bipartite Crossing Number to s-t-Upward Crossing Minimization, which is a minor modification of the original NP-hardness proof by Garey and Johnson [92] that showed hardness of Bipartite Crossing Number. The problem Bipartite Crossing Number is known under a variety of names. Eades et al. refer to it as LEFT OPTIMAL DRAWING [77] and it is also often denoted by 2-LAYER STRAIGHT LINE CROSSING MINIMIZATION [125, 142].

Theorem 5.2 ([92]) S-T-UPWARD CROSSING MINIMIZATION *is NP-hard.*

Layer-free Upward Crossing Minimization

In this section we describe our algorithmic approach to finding a topology of our modified input graph $\widehat{G}$. Note that $\widehat{G}$ may be non-planar and, thus, we cannot directly compute an upward planar embedding.

Since we know that S-T-UPWARD CROSSING MINIMIZATION is NP-complete, we cannot hope to find an efficient algorithm that solves S-T-UPWARD CROSSING MINIMIZATION, unless $\mathcal{P} = \mathsf{NP}$. We are nevertheless interested in finding some reasonably good solution to the problem. For this we now describe an algorithm for which we cannot give any quality guarantee. We build on an algorithmic approach called "layer-free upward crossing minimization" by Chimani et al. [55]. They developed an upward crossing minimization method that works without a prescribed layer assignment of the nodes. Their approach is based upon two steps: (i) Find a large subgraph $U = (\widehat{V}, A')$ of $\widehat{G} = (\widehat{V}, \widehat{A})$ that is upward planar and *feasible*, i. e., the deleted edges can be reinserted such that the resulting drawing is an upward drawing. (ii) Reinsert the deleted edges one by one. For each edge insertion at first a crossing-minimal reinsertion is conducted. However, a crossing-minimal reinsertion of an edge can prohibit the reinsertion of the remaining edges in an upward planar way. In this case, the reinsertion is undone and the edge is reinserted using a simple heuristic. Note that when reinserting edges, we do not count the crossings with edges incident to $\hat{s}$ or $\hat{t}$ since these edges are only used for technical reasons.

After applying this algorithm we obtain an *upward planar representation* of the input graph. An upward planar representation $(\mathcal{U}, \Gamma)$ of a DAG G is an upward planar graph $\mathcal{U} = (V_{\mathcal{U}}, A_{\mathcal{U}})$ together with an upward planar embedding Γ of $\mathcal{U}$. Crossings in the original graph have been replaced by crossing dummies.

upward planar representation

THE ALGORITHM. We omit details regarding the algorithm for computing a feasible upward planar representation by Chimani et al. but give only a sketch of the approach. For details we refer the reader to the work by Chimani et al. [55].

To obtain the upward planar representation, we first compute a feasible upward planar subgraph $U = (\widehat{V}, A')$ of $\widehat{G} = (\widehat{V}, \widehat{A})$ that contains all edges incident to $\hat{s}$ and $\hat{t}$ and an upward planar embedding Γ of U such that $\hat{s}$ and $\hat{t}$ are on the outer face. This is required for the free sources/sink property of the resulting layout. We start with U containing only the incident edges of $\hat{s}$ and $\hat{t}$. The neighbors of $\hat{s}$, $\hat{s}$ itself and $\hat{t}$ are marked as visited. Afterwards, we start directed depth first searches at all neighbours of $\hat{s}$. We add the encountered edges to U if the target has not been marked as visited yet.

Note that, by construction, the subgraph U must be an upward planar s-T-graph. Combining this insight with the *Feasibility Lemma* [55, Lemma 3.6], leads to the conclusion that U is a feasible upward planar subgraph of $\widehat{G}$.

After computing the intermediate feasible upward planar subgraph U, we try to increase the number of edges U contains. To this end, we try to add the edges of $\widehat{A}$ that are not in U one by one and check after each insertion whether it remains a feasible upward planar subgraph. If this check fails we undo the insertion and postpone inserting the edge to the second phase. In the second phase of this approach, we compute the upward planar representation $(\mathcal{U}, \Gamma)$ of $\widehat{G}$. We begin by setting $\mathcal{U} = U$. Then, we add the edges of $\widehat{A}$ that are in $\widehat{G}$ but have not yet been added to $\mathcal{U}$. In this step also the crossing dummies are inserted to $\mathcal{U}$. As in [55] we reinsert the missing edges in a crossing minimal way. If reinserting an edge in a crossing minimal way leads to an embedding which prohibits inserting all remaining edges in an upward planar way, we remove the edge and use a heuristic proposed by Chimani et al. to insert this edge. This heuristic guarantees a valid upward planar embedding but may produce more crossings.

The algorithm that computes a subgraph that is upward and feasible requires $\mathcal{O}(|\widehat{A}|^2)$ time. The algorithm for crossing minimal insertion of an edge a_i requires $\mathcal{O}(|\widehat{V}| + r)$ time, where r is the number of edges that have to be inserted after a_i, whereas the heuristic requires $\mathcal{O}(|\widehat{V}|^2 + r|\widehat{V}|)$ time. The value of r is in $\mathcal{O}(|\widehat{A}|^2)$ since per insertion step at most $O(|\widehat{A}|)$ edges (along with $O(|\widehat{A}|)$ dummy nodes) can be inserted. We need to repeat the insertion step $\mathcal{O}(|\widehat{A}|)$ times, hence the worst-case running time is $\mathcal{O}(|\widehat{A}|^3 \cdot |\widehat{V}|)$. We summarize the results in the following theorem.

Theorem 5.3 *The above algorithm computes the upward planar representation of an s-t-graph $\widehat{G} = (\widehat{V}, \widehat{A})$ in $\mathcal{O}(|\widehat{A}|^3 \cdot |\widehat{V}|)$ time.*

Chimani et al. suggest to randomize the algorithm for the computation of the feasible upward planar subgraph by changing the order in which the edges are considered during the depth first searches, and the order in which the remaining edges are reinserted. Among several runs the crossing minimal upward planar representation is chosen as the overall result. We will take up this idea in the evaluation of our algorithm in Section 5.6.

5.4 SHAPE

In this section we explain the second step of the topology-shape-metrics framework, i. e., how to compute the shape of a layout for a given upward planar representation $(\mathcal{U}, \Gamma)$ of $\widehat{G}$. In this context, shape describes the number of bends on each edge and their bend directions.

The algorithm we present takes up the idea of Biedl and Kant on how to draw a graph with few bends in linear time [28] and adapts it to orthogonal upward drawings with at most four bends per edge. This approach leads to column-based drawings. Since there are at most four bends per edge and ports are at the bottom and the top of the source and target, respectively, an edge can have at most three vertical segments. The column of the first (last) vertical segment is already determined by the column assigned to the source (target, respectively). The position of the middle vertical segment is determined by assigning it to some column. A *column assignment* of a DAG $G = (V, A)$ is a mapping $col \colon V \cup A \to \mathbb{Z}$ that assigns each node $v \in V$ and each edge $e \in A$ a column.

column assignment

Note that a column assignment already induces the shape, i. e., which edges contain bends in which direction. In the metrics step we will realize the column assignment, i. e., we compute a layout that respects the column assignment. Let L be a *valid layout* of a DAG $G = (V, A)$, i. e., L satisfies the local symmetry and four bends per edge constraints. Layout L is a *realization* of a column assignment col, if we can divide the plane into disjoint columns of uniform width and number the columns from left to right such that each node $v \in V$ and each vertical edge $e \in A$ is positioned within column $col(v)$ and $col(e)$, respectively.

Our algorithm positions the nodes beginning from the topmost node to the bottommost node. Hence, we require an order of the nodes for our algorithm that ensures that when placing a node that all its predecessors have been placed already. To do this we can simply compute a topological order for all vertices beginning with $\hat{s}$. This can be done in linear time.

Now, the column assignment algorithm works as follows: The nodes of $\widehat{G}$ are treated according to the computed topological order given by $\hat{s} = v_0, \ldots, v_{n+1} = \hat{t}$. Thus, the algorithm that computes the column assignment starts with the super source $\hat{s}$ and assign it to column 0. Afterwards, we assign the outgoing edges of $\hat{s}$ according to the left-to-right order prescribed by $(\mathcal{U}, \Gamma)$ such that they are distributed evenly to the left and right of column 0 without introducing gaps.

Algorithmus 1 : 4-Bend Column Assignment

```
Input  : Graph $\widehat{G}$, Upward planar representation $(\mathcal{U},\Gamma)$,
         topological order $v_0,\dots,v_{n-1}$ of the nodes of $\widehat{G}$
Output : Column assignment for each node and each edge of $\widehat{G}$
1  $col(v_0) = 0$
2  For $i$ from $1$ to $n-1$ do
3      $I \leftarrow$ column indices assigned to incoming edges of $v_i$
4      $m \leftarrow$ median of $I$
5      $col(v_i) \leftarrow m$
6      shiftLeft($\widehat{G}, m, \lfloor$out-degree$(v_i)/2\rfloor$)
7      shiftRight($\widehat{G}, m, \lfloor($out-degree$(v_i)-1)/2\rfloor$)
8      $j \leftarrow m - \lfloor$out-degree$(v_i)/2\rfloor$
9      For outgoing edge $e$ of $v_i$ from left to right according to $(\mathcal{U},\Gamma)$ do
10         $col(e) \leftarrow j$
11         $j \leftarrow j+1$
```

The invariant of our algorithm is that, when processing vertex v_i, all incoming edges of v_i are already assigned to columns. Then we assign v_i to column m—the column of the median incoming edge—and assign the outgoing edges of v_i to columns according to their left-to-right order induced by $(\mathcal{U},\Gamma)$. However, the columns to the left and right of m are possibly already occupied by other edges. Therefore, we shift all columns left (right) of m to the left (right) such that there are out-degree$(v_i)-1$ empty columns to which the outgoing edges of v_i can be assigned, according to the left-to-right order prescribed by the upward planar representation $(\mathcal{U},\Gamma)$. For a pseudo-code description of this algorithm see Algorithm 1. We now prove that the column assignment computed by our algorithm is realizable.

Theorem 5.4 *The above algorithm computes a realizable column assignment for a given DAG $\widehat{G} = (\widehat{V}, \widehat{A})$ in $\mathcal{O}(|\widehat{A}|)$ time and space.*

Proof. A layout of $\widehat{G}$ can be constructed inductively during the execution of our shape algorithm. Since the x-coordinates are fixed by the column assignment, we only need to deal with the y-coordinates.

Initially, we set the y-coordinate of $\hat{s}$ to 0 and draw the first vertical and horizontal segments belonging to the outgoing edges of $\hat{s}$ as high as possible. When treating v_i, we draw the middle vertical segments of all incoming edges of v_i such that they reach farther down than any other edge or box in the intermediate layout. Then we append the second horizontal segments of these edges and position the box v_i. Afterwards, we draw the first vertical and horizontal segment of the outgoing edges of v_i as high as possible. Thus, no box can be intersected by an edge, and hence, this approach always yields a valid layout.

Since we assign nodes and edges to columns, each edge can contain at most two horizontal edge segments—one spanning from the source's column to the edge's column and one spanning from the edge's column to the target's column. Thus, there are at most four bends per edge. Furthermore, the shape algorithm positions the nodes and edges such that incoming and outgoing edges of each node v_i are symmetrically distributed to the columns left and right of the column assigned to v_i. Thus, the layout satisfies the local symmetry property.

We now turn towards the runtime analysis. The algorithm that computes the topological order of the nodes in $\mathcal{U}$ runs in linear time. The time complexity of the shape algorithm itself depends on the implementation of the two methods SHIFTLEFT and SHIFTRIGHT that shift the already existing columns to the left or to the right. These methods are called

$|V_{\mathcal{U}}|$ times in total. Biedl and Kant suggest an approach that yields an overall runtime of $\mathcal{O}(|\widehat{V}|)$ [28]. They represent the columns by a doubly linked list. Each box and edge has a pointer to the column to which it is assigned. Since the position at which the new columns are going to be inserted are known SHIFTLEFT and SHIFTRIGHT can then be implemented to run in constant time. We also need to compute the median m of the incoming edges for a node. This can be done in $\mathcal{O}(\text{in-degree}(v_i))$ time [30]. We sum up over all iterations:

$$\sum_{i=0}^{n-1} \text{in-degree}(v_i) = |\widehat{A}|$$

Thus, the shape algorithm can be implemented as an $\mathcal{O}(|\widehat{V}| + |\widehat{A}|) = \mathcal{O}(|\widehat{A}|)$-algorithm.

Of course, we need $\mathcal{O}(|\widehat{V}| + |\widehat{A}|)$ space in order to store the assigned columns. Additionally, we need $\mathcal{O}(|\widehat{A}|)$ space in order to store the doubly linked listed that represents the columns. Thus, in total the shape algorithm needs $\mathcal{O}(|\widehat{V}| + |\widehat{A}| + |\widehat{A}|) = \mathcal{O}(|\widehat{A}|)$ space. □

Note that if the upward planar representation $(\mathcal{U}, \Gamma)$ contains no crossing dummies, then the realization constructed in the proof of Theorem 5.4 is upward planar as well.

Observation 5.1 *The realizations of the column assignments of an upward planar representation $(\mathcal{U}, \Gamma)$ that contains no crossing dummies are upward planar layouts.*

As we already have mentioned, during the construction of the column assignment, we only respect the left-to-right order of the outgoing edges at each node of $\widehat{G}$ that is prescribed by $(\mathcal{U}, \Gamma)$. We do not consider the crossing dummies in $\mathcal{U}$. Thereby, we relax the topology that we computed in the first step. Because of the relaxation of the topology, the final layout can have a higher number of crossings than the topology. On the other hand, this lets us focus on producing more compact layouts. Since we still maintain parts of the result of the first step, which minimized the number of crossings, we expect only a small increase in the number of crossings. Moreover, the crossings in our drawings are only right-angle crossings, and thus do not hinder readability that much.

5.5 METRICS

In the last phase of the approach, we compute the final coordinates of boxes and edges while minimizing the total edge length. The topology that we originally computed in the first step has no *direct* influence *anymore*. Instead, we only rely on the column assignment computed by the shape algorithm.

The metrics step consists mainly of two substeps. First, we focus on minimizing the vertical edge length. Since we can show that this problem is NP-complete, we suggest a simple heuristic in order to solve it. Afterwards, we minimize the horizontal edge length. Note that all operations are performed with respect to the columns so that the final result is a column-based layout.

5.5.1 *Vertical Edge Length Minimization*

We begin by showing that finding a realization of a column-assignment with minimum total vertical edge length (or, equivalently, minimum total height) is NP-complete. Since we are still interested in finding a solution (albeit possibly a not optimal solution) we propose after the NP-completeness proof a simple greedy heuristic to tackle this problem.

5.5.1.1 *Computational Complexity*

We call the problem of finding a realization of a column assignment with minimum total vertical edge length the VERTICAL EDGE LENGTH MINIMIZATION problem. The formal definition of the problem is as follows.

Instance: An s-t-graph $\widehat{G} = (\widehat{V}, \widehat{A})$, a column assignment for each node in $\widehat{V}$ and each edge in $\widehat{A}$, a set of spacing constraints and an integer $k \geq 0$.
Question: Is it possible to assign y-coordinates to the boxes and edge bends of $\widehat{G}$ such that the resulting layout is valid and the total vertical edge length is at most k?

We prove NP-completeness of Vertical Edge Length Minimization by reduction from 3-Partition, which is defined as follows [91].

Instance: A finite set $A = \{a_1, \dots, a_{3m}\}$ of $3m$ elements, a bound $B \in \mathbb{Z}^+$ and a "size" $s(a) \in \mathbb{Z}^+$ for each $a \in A$, such that each $s(a)$ satisfies $B/4 < s(a) < B/2$ and such that the equation $\sum_{a \in A} s(a) = mB$ holds.
Question: Can A be partitioned into m disjoint sets $S_1, S_2, \dots, S_m$ such that for $1 \leq i \leq m$ the equation $\sum_{a \in S_i} s(a) = B$ holds? (Note that the above constraints on the item sizes imply that every such S_i must contain exactly three elements from A.)

Garey and Johnson proved that this problem is NP-complete in the strong sense. This means that 3-Partition remains NP-complete if the numeric values encoded in the input data are polynomially bounded by the length of the input.

In the following, we describe how to transform a 3-Partition instance to a Vertical Edge Length Minimization instance. This means we need to describe how to construct an s-t-graph and a corresponding column assignment of all vertices and edges of the s-t-graph in polynomial-time such that a solution of Vertical Edge Length Minimization of this instance induces a solution of 3-Partition.

The key idea behind our reduction is to use a single column that we divide into m areas, which we call *cells*, and we want that the each of the m cells corresponds to one of the m sets in a solution to a 3-Partition instance. The elements in A are represented in our reduction by nodes in the same column as the cells and whose heights is their "size" in the 3-Partition instance. Ultimately, we want that in an optimal solution of Vertical Edge Length Minimization (in our reduction) from a 3-Partition instance $\mathcal{I}$ there are exactly three nodes in each cell whose heights add up to exactly B if and only if $\mathcal{I}$ is a Yes-instance.

For ease of argumentation we set all minimum spacing constraints to 0, but we note that the reduction itself does not strictly require that all minimum spacing constraints are 0, and only minor modifications are necessary for the reduction still to work for arbitrary spacing constraints. However, setting the minimum spacings constraints to 0 has the unfortunate side-effect that the drawings are not visually appealing. For the figures in this section we assume that all minimum spacing constraints are set to a small positive value.

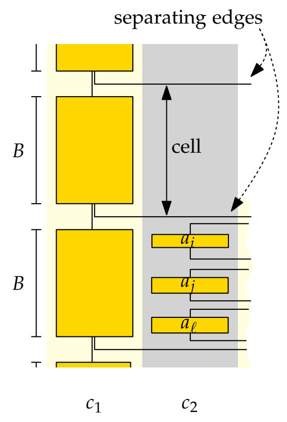


Figure 5.3: Sketch of the cells.

In our reduction all nodes are placed within $|A| + 3$ adjacent columns, which we denote by $c_1, \dots, c_{|A|+3}$. All cells will be inside column c_2. Unless stated otherwise, nodes in our graph have the same height h, where h is a small but fixed value.

We now discuss the construction of the cells. For each cell we add two nodes with height B to our graph, one assigned to column c_1 and one assigned to the rightmost column $c_{|A|+3}$ (we need the columns between c_1 and $c_{|A|+3}$ for additional nodes, which we describe later). For the i-th cell we denote the node placed in column c_1 by ℓ_i, and the node placed in column $c_{|A|+3}$ by r_i. Now, we divide column c_2 by adding a directed edge (ℓ_i, r_{i+1}) for every i, $0 \leq i \leq m + 1$. We call these edges *separating edges*; for a sketch see Figures 5.3 and 5.4. Note that we require nodes ℓ_i and r_i for $i = 0, \dots, m + 1$ in order to achieve that each of the m cell is bounded by a separating edge from above and from below. Finally, we connect the nodes $\ell_0, \dots, \ell_{m+1}$ such that they form a directed path from ℓ_0 to

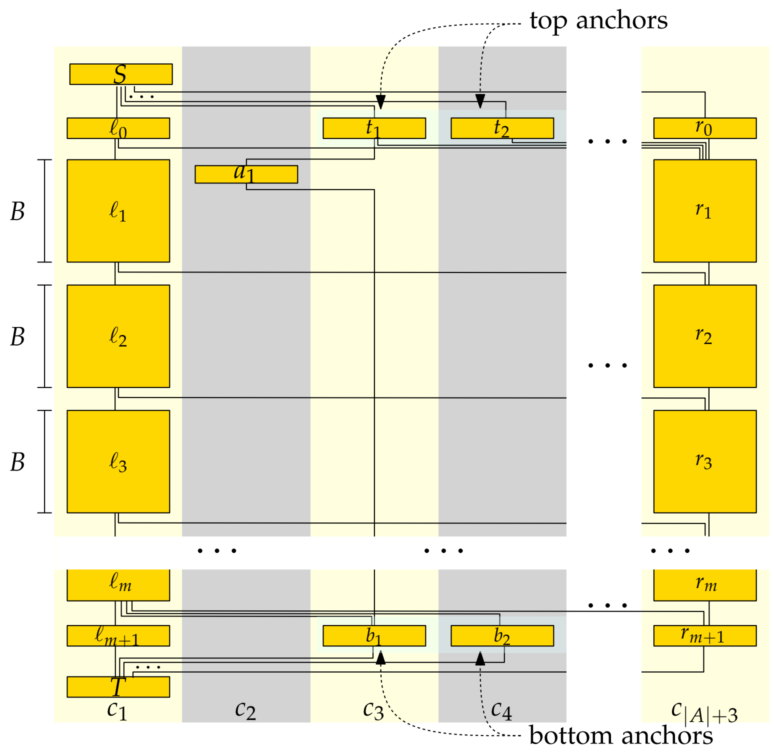


Figure 5.4: Sketch of the Hardness Proof

ℓ_{m+1} and do the same for the nodes $r_0, \dots, r_{m+1}$. Since the topmost and bottommost nodes of each path do not themselves span a cell, we set their heights to h.

For each $a_i \in A$ we add a node v_i, which we refer to as *element node*, with height $s(a_i)$ to our graph. Since the cells are meant to correspond to the sets in the 3-PARTITION instance, we must be able to position any of the element nodes v_i in any of the cells. On the other hand, we must be sure that placing a node outside of the cells increases the total vertical edge length. To ensure this we make use of *anchor nodes*. For each element node v_i we add two anchor nodes, a *top anchor* t_i and a *bottom anchor* b_i, both assigned to column c_{i+2}, along with directed edges (t_i, v_i) and (v_i, b_i) to our graph. To ensure that the anchor nodes themselves stay above/below the cells we add for each top anchor t_i a directed edge (t_i, r_1) and for each bottom anchor a directed edge (ℓ_m, b_i).

Since the top (bottom) anchors stay above (below) the cells, moving the element nodes upwards or downwards is cost neutral, because the sum of the vertical edge lengths on the two edges incident to the anchors remains unchanged. Further, note that there are no edges between the different element nodes, i. e., their vertical ordering is not constrained. Since a element node v_i has height $s(a_i)$, finding three nodes whose total height sums up to B means we can place them in the same cell without increasing the total vertical edge length.

We are nearly finished with the construction of our graph, the only thing left is to ensure that the graph we constructed is an s-t-graph. This can easily be achieved as follows. Add two nodes S and T to the graph, assign them both to column c_1 and add directed edges from S to all top anchors, as well as ℓ_0 and r_0, and add directed edges from all bottom anchors, as well as ℓ_{m+1} and r_{m+1} to T. Further, to obtain a complete column assignment we assign each edge to the column of its source. We have now finished the construction of the graph and the column assignment. It remains to compute the total vertical edge length k in an optimal solution to our VERTICAL EDGE LENGTH MINIMIZATION instance if the corresponding 3-PARTITION is solvable. Since we have chosen all spacing constraints to be 0, and each element a_i contributes $mB - s(a_i)$ to the vertical edge length, the minimum possible vertical edge length of the edges incident to the element nodes is $3m^2B - mB$. All remaining edges may have vertical edge length 0. We set $k = 3m^2B - mB$.

CORRECTNESS. We now show, that the 3-PARTITION instance has a solution if and only if the VERTICAL EDGE LENGTH MINIMIZATION instance is solvable. Obviously, if the 3-PARTITION instance has a solution, then it can be transformed to a solution of the VERTICAL EDGE LENGTH MINIMIZATION instance by vertically ordering the element nodes. If the VERTICAL EDGE LENGTH MINIMIZATION instance is solvable, we need to show that the 3-PARTITION instance has a solution as well. If the solution of VERTICAL EDGE LENGTH MINIMIZATION contains nodes that are not placed inside one of the cells, then this increases the total vertical edge length since there is no possibility of decreasing the total edge length elsewhere. Further, should there be more than three nodes in one cell, then, the sum of the heights of those nodes must add up to more than B (recall that $B/4 < s(a_i) < B/2$). Since the minimum height of each cell is B this increase in vertical edge length cannot be compensated elsewhere. Hence, we can conclude that in an optimal solution to VERTICAL EDGE LENGTH MINIMIZATION of a Yes-instance of 3-PARTITION there are exactly three nodes in each cell. This directly induces a solution to the corresponding 3-PARTITION instance.

Hence, we can conclude that VERTICAL EDGE LENGTH MINIMIZATION is NP-hard. Since it is easy to see that VERTICAL EDGE LENGTH MINIMIZATION is contained in NP, VERTICAL EDGE LENGTH MINIMIZATION is NP-complete. We summarize this result in the following theorem.

Theorem 5.5 VERTICAL EDGE LENGTH MINIMIZATION *is NP-complete.*

In the following we suggest a three-step approach for how to cope with the problem of vertical edge length minimization.

5.5.1.2 *Greedy Algorithm*

Since VERTICAL EDGE LENGTH MINIMIZATION is NP-complete, we propose a greedy algorithm to minimize the vertical edge length. Recall that we want each node to have as many predecessors as possible vertically aligned at their bottom. We integrate this requirement into our vertical edge length minimization approach. In the first step we determine a partition of the nodes into sets which we call *groups*, each of which contains only nodes that can be aligned at their bottom. Hence, we need to be careful which nodes belong to the same group. It is easy to see that there must not be a directed path between any two nodes belonging to the same group. Further, it is necessary that the groups can be topologically ordered, i. e., the graph that we obtain by contracting each group to a single node must remain acyclic. In the second step, we compute a reverse topological order of these groups and finally, in the third step, we position the groups according to this order.

GROUPING.. We call a set of pairwise disjoint groups of nodes $g_1, \ldots, g_k$ *feasible* if the graph that is obtained from $\widehat{G}$ by contracting each group to a single node is acyclic. A node $s \in \widehat{V}$ is an *immediate predecessor* of another node $s' \in \widehat{V}$ if there is an edge $(s, s') \in \widehat{A}$, and there is no other directed path from s to s' in $\widehat{G}$. We denote the set of immediate predecessors of a node s that have not been assigned to a group by PRED(s).

The algorithm works in several iterations, each of which greedily computes a feasible set groups. We describe the ith iteration. If all nodes have been assigned to a group, the algorithm terminates. Otherwise, the algorithm creates a group g_i that contains an arbitrary node of $\widehat{G}$ that has not yet been assigned to a group. The algorithm then tries to increase the size of g_i as much as possible by determining nodes that are potential candidates for inclusion in g_i. Because we want the nodes in each group to be vertically aligned at their bottom, potential candidates are nodes that share the same successor with a node already contained in g_i. More specifically, the algorithm determines for each node v in g the set SUCC of their successors. For each node $s \in$ SUCC the algorithm determines the set of immediate predecessors PRED(s) that are not yet assigned to a group, and whether adding PRED(s) to g_i yields a feasible set of groups, i.e., whether $g_1, \dots, g_{i-1}, g_i \cup \text{PRED}(s)$ is feasible. If this is the case, then we replace g_i by $g_i \cup \text{PRED}(s)$; otherwise the algorithm continues with the next successor $s' \in$ SUCC. Note that adding new nodes to g_i can require adding new nodes to SUCC. When no new nodes can be added to g_i in this way, the ith iterations finishes. For a pseudo-code description of this algorithm see Algorithm 2.

Algorithmus 2 : Grouping Algorithm

```
Input   : Graph $\widehat{G} = (\widehat{V}, \widehat{A})$
Output  : Groups $g_1 \dot{\cup} g_2 \dot{\cup} \dots \dot{\cup} g_k = \widehat{V}$ of the nodes in $\widehat{G}$
1  For $1 \leq i \leq n$ do
2      $g_i = \emptyset$
3  $i \leftarrow 1$
4  while $\exists n \in \widehat{V} : n \notin g_j$ for $1 \leq j < i$ do
5      Insert $v$ into $g_i$
6      SUCC $\leftarrow$ successors of $v$
7      while SUCC $\neq \emptyset$ do
8          For $s \in$ SUCC with PRED($s$) do
9              If $g_1, \dots, g_{i-1}, g_i \cup$ PRED($s$) is feasible
10                 $g_i = g_i \cup$ PRED($s$)
11         SUCC $\leftarrow$ successors of nodes added to $g_i$ in this iteration that have not been
           treated yet
12     $i \leftarrow i + 1$
```

Lemma 5.1 *The grouping algorithm can be implemented to run in $\mathcal{O}(|\widehat{V}|^2 \cdot |\widehat{A}|)$ time and $\mathcal{O}(|\widehat{V}| + |\widehat{A}|)$ space.*

Proof. In each iteration i of the algorithm, the key operation is to determine whether $g_1, \dots, g_{i-1}, g_i \cup \text{PRED}(s)$ is a feasible set of groups. To check this, we maintain the DAG that is obtained from $\widehat{G}$ by contracting each group g_i to a single node G_i. Including PRED(s) into g_i introduces a cycle in this graph if and only if there is a path from a node in PRED(s) to G_i and a path from G_i to a node in PRED(s). This can be easily checked with two breadth-first searches. Note that the contracted graph has size $\mathcal{O}(|\widehat{V}| + |\widehat{A}|)$, and hence the test can be performed in time $\mathcal{O}(|\widehat{V}| + |\widehat{A}|)$. When we include a set PRED(s) into g_i, we immediately contract the nodes in PRED(s) into G_i. Clearly such a contraction is performed only once per node, and hence this takes total time $\mathcal{O}(|\widehat{V}| + |\widehat{A}|)$ over all iterations.

The running time of the algorithm is dominated by lines 4-11 by the two nested loops and the feasibility check. The outer loop can be executed $\mathcal{O}(|\widehat{V}|)$ times since there are $\mathcal{O}(|\widehat{V}|)$ groups. We can also bound the number of iterations for the inner while loop by $\mathcal{O}(|\widehat{V}|)$ if we can bound the number of elements that are added to SUCC by $\mathcal{O}(|\widehat{V}|)$. We can guarantee this if we store during each iteration for each node if it has already been added to SUCC before. Then, before adding a node to SUCC (l. 10) we first check whether we have added the node before already. If this is the case we do not add it to SUCC again. Finally, for the feasibility check we require two breadth-first searches that require $\mathcal{O}(|\widehat{V}| + |\widehat{A}|)$ time each. Since $\mathcal{O}(|\widehat{V}|) \subseteq \mathcal{O}(|\widehat{A}|)$ the running time of our algorithm is $\mathcal{O}(|\widehat{V}|^2 \cdot |\widehat{A}|)$. □

COMPUTING THE ORDER. After computing the grouping, we compute an order of the groups in the second step. Since we draw the layout bottom-up in the third step, when positioning a group g_i, all successors of nodes in g_i need to be already positioned. To do this we contract each group and sort them topologically from bottom to top. Recall that our algorithm ensures that when adding new nodes to a group the group remains feasible, i. e., there is no directed path in the graph from one node in the group to another in the same group. This directly implies that we can always order the groups topologically.

COORDINATE ASSIGNMENT. In the last step of the greedy approach the actual coordinate assignment takes place. We treat the groups according to the order computed in the previous step. The y-coordinates of the ports and the bends of an edge are computed in two parts. When the target of an edge is positioned, we compute the y-coordinates of the target port and the first two bends. The first bend is in the target's column, whereas the other one is in the column assigned to the edge. The two last bends and the source port are computed, when its source is positioned.

First, we compute the smallest possible y-coordinate $\hat{y}$ for the lower boundary of the nodes in g_i by setting it to the maximum of the smallest possible y-coordinate for each node $v_j \in g_i$. When computing this y-coordinate for a node $v_j \in g_i$, we need to consider that we have not yet routed the second part of the edges whose targets are in g_i. For this, we need to compute the smallest possible y-coordinate for the horizontal edge segment of the edges we still need to draw. In particular, we need to find the smallest possible y-coordinate such that all minimum spacing constraints are respected. This can be done straightforwardly.

After determining $\hat{y}$, we position the boxes in g_i such that their lower boundary is aligned at $\hat{y}$ and draw the second part of the outgoing edges. We draw the horizontal segments as high as possible. Afterwards, we draw the incoming edges as low as possible. To this end, we process the nodes $v_j \in g_i$ in an arbitrary ordering and draw their incoming edges such that all spacing constraints hold.

The running time of this step is $\mathcal{O}(|\widehat{V}| + |\widehat{A}| \cdot b)$, where b is the number of columns used by the assignment. Note that since every column contains a vertex it follows that b is in $\mathcal{O}(|\widehat{V}|)$. Each node in $\widehat{V}$ is considered only once, i. e., when it is positioned. The edges are treated twice. Once when the target is positioned and its first part is drawn and a second time when its source is positioned, i. e., the second part of the edge is drawn. Both times the smallest possible y-coordinate needs to be determined, which requires as many operations as the drawn horizontal segment spans columns. We simply bound this number of columns by $|\widehat{V}|$.

Theorem 5.6 *The vertical edge length minimization heuristic can be implemented to run in $\mathcal{O}(|\widehat{V}|^2 \cdot |\widehat{A}|)$ time and $\mathcal{O}(|\widehat{V}| + |\widehat{A}|)$ space.*

Proof. The computation of the groups in step one takes $\mathcal{O}(|\widehat{V}|^2 \cdot |\widehat{A}|)$ time, afterwards the groups can be ordered in $\mathcal{O}(|\widehat{V}| + |\widehat{A}|)$ time. For the final y-coordinate assignment we need $\mathcal{O}(|\widehat{V}| + |\widehat{A}| \cdot |\widehat{V}|)$ time. Thus, the greedy approach runs in $\mathcal{O}(|\widehat{V}|^2 \cdot |\widehat{A}|)$ time in total.

For the computation of the groups $\mathcal{O}(|\widehat{V}| + |\widehat{A}|)$ space is required. Furthermore, we need $\mathcal{O}(|\widehat{V}| + |\widehat{A}|)$ space for storing the y-coordinates assigned to the boxes and the edge ports and bends. Additionally, we store for each of the at most $\widehat{V}$ columns the highest object that is already drawn. Thus, in total our algorithm requires $\mathcal{O}(|\widehat{V}| + |\widehat{A}|)$ space. □

5.5.2 *Horizontal Edge Length Minimization*

There has been some research into improvement of orthogonal drawings which is related to the approaches used in this section. Six et al. [171] discuss several refinement steps for orthogonal drawings. They propose heuristics to remove U-Turns (which we refer to as *bows*; see Figure 5.5), superfluous bends, self-crossings, and several other unappealing structures. However, we encounter, due to the way our algorithm works, only few of the problems described in this. The only structure we remove are bows, by using a simple linear time algorithm. However, there are other examples of structures that are not visually appealing

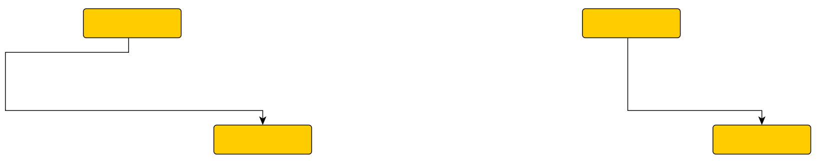


Figure 5.5: An edge that is a bow (left) and the edge after bow reduction (right).

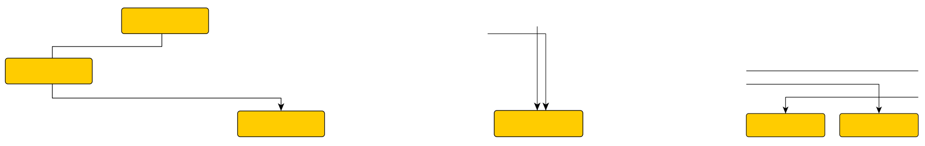

(a) (b) (c)

Figure 5.6: Visually unappealing parts of a drawing which can be resolved in a post-processing step.

and that can easily be resolved but which we did not consider for the implementation; see Figure 5.6 for some examples.

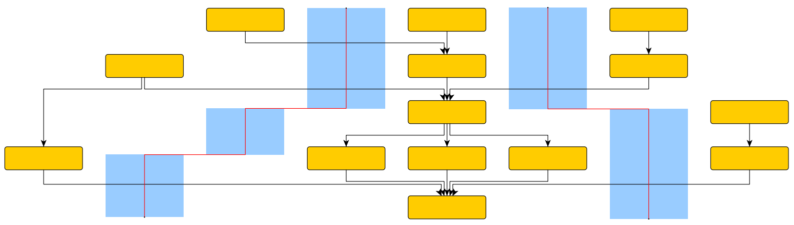
In the second step, we employ a heuristic approach to minimize the width of the whole layout. Due to the shifting in each iteration of the shape algorithm, we introduce new columns. Some of them are necessary while others will be partly empty in the final layout and may be removed while maintaining a valid layout. Thus, the final layout is possibly wider than actually necessary. In Figure 5.7(a) we depict a layout that can be compacted by two columns. Our approach to width compaction is column based, i. e., the resulting layout again consists of disjoint columns.

Our approach is somewhat similar to the *moving* part of the 4M-algorithm proposed to improve orthogonal drawings [88]. For this moving part of the 4M-algorithm a *moving line* is defined which is directed through the drawing and used to identify empty space in the drawing that can be removed safely. However, the approach requires the node heights to be a multiple of a certain base value and the nodes must be aligned on a grid. We perform width compaction along so-called *compaction paths*. A compaction path is an axis-parallel y-monotone path passing through a layout from the top to the bottom. It may cut horizontal edge segments but neither vertical edge segments nor nodes. Furthermore, its vertical segments need to run within columns such that no vertical edge segment is allowed to run through the same part of these columns. Each compaction along a compaction path decreases the number of columns by one.

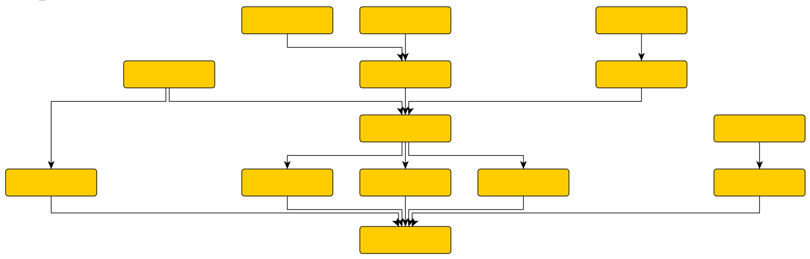
compaction paths

In Figure 5.7(b) we illustrate how compaction along a compaction path is performed. First, we split the path into its vertical segments. Then we delete the y-range of a column that contains a vertical path segment. Thus, in total we gain one column of free space. We move everything that has been to the right of the compaction path one column to the left and, thereby, fill the free space again. Thus, in total the number of columns is decreased by one.

A seemingly obvious approach to compute a maximum set of compaction paths would be by using a flow algorithm. Traditionally, flow algorithms have been used in graph drawing to solve compaction problems [93, 177]. However, in our case there are examples of compaction paths where each path by itself is valid, but compacting two such paths at the same time yields a layout where an edge-edge spacing constraint is violated; see Figure 5.8. Although it might be possible to avoid this problem when using flow networks by increasing the size of the drawing, the flow network itself is very complicated (see [71] for a description) and a simpler approach appears to be reasonable. We propose an approach

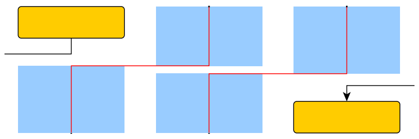


(a) A layout that can be compacted by two columns and two compaction paths.

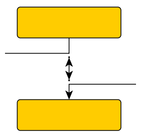


(b) How to perform compaction along the left compaction path.

Figure 5.7: Compaction paths and compaction along a compaction path.



(a) The set of compaction paths along which we compact. Both compaction paths are valid.



(b) After the compaction the spacing constraint between the two edges can be violated.

Figure 5.8: An invalid set of compaction paths.

that iteratively removes columns instead of removing multiple columns at once. This helps us to avoid the problem stated above.

In the following, we describe an approach that computes a cardinality-maximal *valid* set of compaction paths using right-first search. We call a set $\mathcal{P}$ of compaction paths valid if there is no pair of paths P_i, P_j for $i \neq j$ in $\mathcal{P} = \{P_1, \ldots, P_k\}$ that cross each other, or overlap. Since the paths do not cross or overlap, we can order them from right to left as $P_1, \ldots, P_k$. We also require that for $i = 1, \ldots, k-1$ there is no path P' distinct from $P_1, \ldots, P_{i+1}$ whose nodes lie to the right of, or on P_{i+1} such that $\{P_1, \ldots, P_i, P'\}$ is a valid set of compaction paths. For the first path P_1 this means that no subpath of it can be replaced by a path that is right of P_1 such that the resulting path is a valid compaction path. We call a valid set of compaction paths that has these properties *rightmost*. In Figure 5.9 we depict a rightmost valid set of compaction paths for the example of Figure 5.7(a). Note that since all paths are directed downwards, a right-first search yields paths that are as left as possible in the drawing.

It remains to argue that by using right-first search we actually find a cardinality-maximal valid set of compaction paths. In the following we show that we can transform any valid set of compaction paths $\mathcal{P} = \{P_1, \ldots, P_k\}$ to a rightmost set of compaction paths of the same cardinality. First, we remove all crossings by swapping the suffixes of any two paths in the set that cross. Note that by doing this we cannot introduce a new crossing. Since the resulting paths $P_1, \ldots, P_k$ do not cross, we can order them from right to left. We treat the paths P_i in increasing order of their indices (from right to left) and try to move each path rightwards. More specifically, for each treated path P_ℓ we a replace a subpath $(p_i, \dots, p_j)$ of it with a path $(q_1, \dots, q_l)$ that is right of P_ℓ and does not overlap or cross a path $P_{\ell'}$ in the set $\mathcal{P}$ with $\ell' < \ell$.

After these normalization steps, we obtain a rightmost valid set of compaction paths. Note that we do not change the number of paths during the normalization. Thus, there exists a unique maximum set of compaction paths that is rightmost, which can be found by the right-first approach we present in the following paragraph.

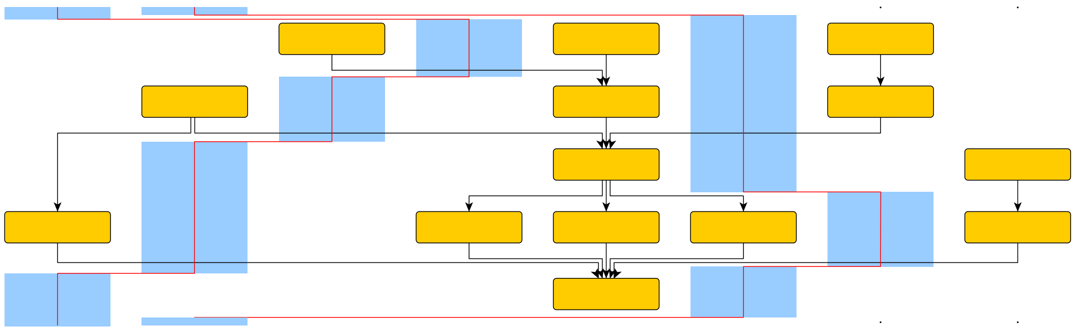


Figure 5.9: A rightmost valid set of compaction paths.

THE ALGORITHM. In this paragraph we present the algorithm that performs the width compaction. Before we run the algorithm, we remove the super source $\hat{s}$ and the super sink $\hat{t}$. As already mentioned, this algorithm finds compaction paths by using a right-first search. Initially, we build up the compaction network in which we search for compaction paths using the right-first search. After we found a compaction path, we need to slightly modify the compaction network in order to ensure that the computed set of compaction paths is valid.

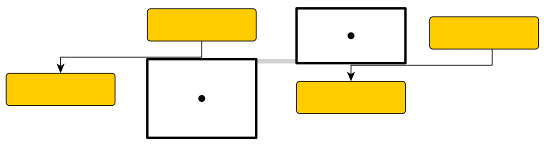
First, we describe the initialization of the graph $\mathcal{D}$, which we refer to as the *compaction network*. The compaction network depends on empty regions in the columns, i. e., we need to find the empty rectangles in each column. Therefore, we create a list per column which will contain the boxes and edge segments that are contained in the corresponding column. Afterwards, we iterate over all boxes and edges. We add the boxes to the list of the column to which they are assigned. The edges are treated more fine-grained. All three vertical segments are added to the list of the corresponding column. Furthermore, we add the horizontal segments to all columns between the source's column and the edge's column and the edge's column and the target's column, respectively. Thus, in total we have $\mathcal{O}(|\widehat{V}| + |\widehat{A}| \cdot b)$ entries in the lists. Since $\widehat{G}$ is connected and, therefore, $\mathcal{O}(|\widehat{V}|) \subseteq \mathcal{O}(|\widehat{A}|)$, we can simplify the number of entries to $\mathcal{O}(|\widehat{A}| \cdot b)$. Each of these entries is associated with two y-coordinates that determine the y-range that is occupied by the box or edge segment, respectively. We sort these entries according to their y-coordinates in $\mathcal{O}(|\widehat{A}| \cdot b \cdot \log(|\widehat{A}| \cdot b))$ time.

compaction network

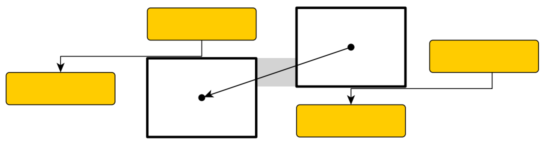
Afterwards, we detect the free rectangles in the columns, i. e., the gap between two entries in the list whose y-ranges do not touch. Furthermore, we assume that on the top and on the bottom of each column is a large free rectangle. For each free rectangle we add a node to $\mathcal{D}$. For ease of notation we identify each node of $\mathcal{D}$ with its corresponding free rectangle.

Since we have at most $\mathcal{O}(|\widehat{A}| \cdot b)$ entries that bound the free rectangles, there can be at most $\mathcal{O}(|\widehat{A}| \cdot b)$ rectangles, i. e., at most $\mathcal{O}(|\widehat{A}| \cdot b)$ nodes in $\mathcal{D}$.

We create the edges of $\mathcal{D}$ in two ways: For two rectangles in the same column, we add an edge between them if they are only separated by a single horizontal edge segment. Furthermore, we create edges between rectangles in neighboring columns if their y-ranges

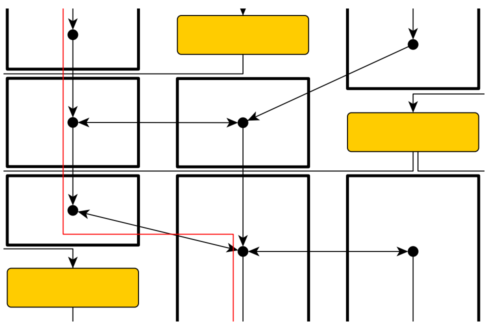


(a) Overlap of both boxes is smaller than $s^{\text{edge}}_{\text{edge}}$.

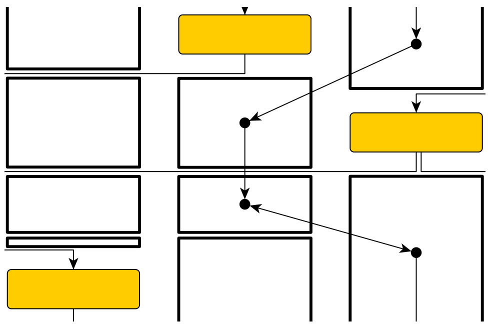


(b) Overlap of both boxes is larger than $s^{\text{edge}}_{\text{edge}}$.

Figure 5.10: Edges between overlapping boxes of neighboring columns.



(a) The compaction network $\mathcal{D}$ and a compaction path p.



(b) The modified network after the compaction along p.

Figure 5.11: The compaction network $\mathcal{D}$ and how it is modified.

overlap. Consider two overlapping rectangles in neighboring columns as depicted in Figure 5.10(b). If a compaction path p runs through these rectangles as shown in Figure 5.10(a), compacting along p would move the right box under the left one. Then the distance between the two edges needs to be at least $s^{\text{edge}}_{\text{edge}}$ or $\hat{s}^{\text{edge}}_{\text{edge}}$, respectively. Note that this distance after the compaction equals the overlap of the y-ranges of the two rectangles. Therefore, we only add an edge from the right to the left rectangle, if the overlap of the y-ranges of the two rectangles is at least the minimum spacing $s^{\text{edge}}_{\text{edge}}$ or $\hat{s}^{\text{edge}}_{\text{edge}}$, respectively; see Figure 5.10(a).

As the final step of the construction of $\mathcal{D}$, we add two nodes to $\mathcal{D}$ which we denote by $\bar{s}$ and $\bar{t}$. We connect $\bar{s}$ to all rectangles on the top of the columns and connect all rectangles on the bottom of the columns with $\bar{t}$.

Obviously, $\mathcal{D}$ is a planar graph, i. e., it has at most $\mathcal{O}(|\widehat{A}| \cdot b)$ edges. However, the number of rectangle pairs we need to consider to find all edges is small. In fact this number is in $\mathcal{O}(|\widehat{A}| \cdot b)$ since by construction edges are either completely contained in a column or connect nodes in two neighboring columns. Since we have sorted the nodes of $\mathcal{D}$ already in a previous step, we find all edges of $\mathcal{D}$ in $\mathcal{O}(|\widehat{A}| \cdot b)$ with a simple sweep line algorithm from top to bottom. In Figure 5.11(a) we depict a clipping of a layout and the corresponding compaction network $\mathcal{D}$.

We use the compaction network $\mathcal{D}$ in order to find compaction paths using a right-first depth-first search starting at $\bar{s}$. Since we need to ensure that the computed compaction path is y-monotone, we always keep track of the current y-coordinate. We are only allowed to move to a horizontally neighboring rectangle if this requires no upward movement of the compaction path. If the search reaches $\bar{t}$, we have found a compaction path.

After a compaction path has been found, we need to slightly modify the compaction network. The part of the compaction network that is right of the compaction path cannot be part of a further compaction path, because we use a right-first search. Therefore, we do not need to consider this part. However, we need to treat the part through which the compaction path cuts (see Figure 5.11): We split rectangles that contain (a part of) a vertical segment of the compaction path such that there is one rectangle that is cut from top to bottom and one or two rectangles that are not cut by the compaction path. Rectangles that are crossed horizontally are split into two rectangles as well.

For all rectangles that are cut from top to bottom by a vertical path segment we delete the corresponding nodes in $\mathcal{D}$. Afterwards, we need to create edges between the newly created rectangles on the left side of the compaction path and their surrounding rectangles. In Figure 5.11(b) we illustrate the modification of $\mathcal{D}$ after the compaction along a compaction path. After the modification of $\mathcal{D}$ we continue the right-first search in order to find more compaction paths. The set of compaction paths we compute is (i) rightmost, because we use a right-first search and (ii) valid due to the construction of $\mathcal{D}$ and its modifications.

Theorem 5.7 *The width compaction algorithm can be implemented to run in $\mathcal{O}(|\widehat{A}| \cdot b^2 \cdot (\log(|\widehat{V}|) + b))$ time and $\mathcal{O}(|\widehat{A}| \cdot b^2)$ space.*

Proof. We already argued that $\mathcal{D}$ has at most $\mathcal{O}(|\widehat{A}| \cdot b)$ nodes and it is planar. However, the size of $\mathcal{D}$ can increase due to the modifications after the compaction along a path. Recall that we are dealing with rightmost compaction paths. Thus, each horizontal segment of a compaction path coincides with the upper or lower boundary of a rectangle. Otherwise, we can replace a subpath of the compaction path with a path that is right of it, i. e., the compaction path was not rightmost. We now estimate the number of rectangles we need to create at most.

Note that a compaction path cannot split a rectangle at any point, but rather it can only split it at a y-coordinate at which a compaction path leaves a column and enters a new one. These y-coordinates are the y-coordinates of the upper and lower boundaries of nodes in $\mathcal{D}$. This insight allows us to find an upper bound on the number of nodes the compaction network can have. As we have argued, initially the number of nodes in $\mathcal{D}$ is in $\mathcal{O}(|\widehat{A}| \cdot b)$. To obtain a worst-case upper bound for the number of new nodes, assume we split all rectangles at the y-coordinates of the upper and lower boundary of a single node. This can create at most two additional rectangles per column which means $\mathcal{O}(b)$ additional rectangles in total. When we repeat this for all of the original nodes in $\mathcal{D}$ this leads to a total of $\mathcal{O}(|\widehat{A}| \cdot b^2)$ nodes in $\mathcal{D}$. Note that we do not need to do this for the newly created nodes, because the y-coordinate of either their upper or lower boundary was already used to cut other nodes. Since $\mathcal{D}$ remains planar, the number of edges is also in $\mathcal{O}(|\widehat{A}| \cdot b^2)$. In order to analyze the time complexity of the right-first depth-first search that operates on a changing graph, we assume that it is directly executed on the final graph $\mathcal{D}$. Thus, it runs in $\mathcal{O}(|\widehat{A}| \cdot b^2)$ time.

When all compaction paths are found, we need to compact along them. Compacting along one path can easily be performed in $\mathcal{O}(|\widehat{V}| + |\widehat{A}| \cdot b^2) = \mathcal{O}(|\widehat{A}| \cdot b^2)$ time. Thus, compacting along all paths needs $\mathcal{O}(|\widehat{A}| \cdot b^3)$ time.

Summing up the running time of the initial construction of $\mathcal{D}$, the computation of the compaction paths and the compaction itself, we get $\mathcal{O}(|\widehat{A}| \cdot b \cdot (\log(|\widehat{A}| \cdot b) + b^2))$ as total runtime. Since we deal with simple graphs, $|\widehat{A}| \leq |\widehat{V}|^2$ holds. Thus, we can further simplify the time complexity to $\mathcal{O}(|\widehat{A}| \cdot b^2 \cdot (\log(|\widehat{V}|) + b))$.

We need $\mathcal{O}(|\widehat{V}| + |\widehat{A}|)$ space to store the column assignment, but this term is dominated by the potential size of $\mathcal{D}$ which is in $\mathcal{O}(|\widehat{A}| \cdot b^2)$. □

Note that in the case that the node sizes do not differ too much, and should all of them be integer, the time complexity of the algorithm can be reduced to $\mathcal{O}(|\widehat{A}| \cdot b^3)$ by employing a linear-time sorting algorithm for this case.

We have now all results required to prove our main result which we stated in Theorem 5.1, and which we repeat here for the reader's convenience. The proof follows by combining the main theorems of each section, namely, Theorems 5.3, 5.4, 5.6, and 5.7. The running time is dominated by the first step of our approach (see Theorem 5.3), and the space consumption is dominated by the last step (see Theorem 5.7).

Theorem 5.1. *For a given connected directed acyclic graph $G = (V, A)$ a locally-symmetric, column-based layout with at most four bends per edge can be computed in $\mathcal{O}(|A|^3|V|)$ time and $\mathcal{O}(|A||V|^2)$ space.*

5.6 EVALUATION

The drawing style used here stems from a particular application for drawings of so-called argument maps. Argument maps present the arguments given in a debate or a book together with two binary relations among them, i. e., support and attack. They originate from argumentation theory but are used in many more fields like philosophy and politics. For detailed background information about argument maps we refer the reader to [25, 166]. As we mentioned at the beginning of Section 5.3, for layouts of argument maps, we have the additional constraints of *free sources and sinks*. Recall that, by this, we denote the property that above a source and below a sink no other box is positioned. We can enforce this constraint by not removing the edges incident to $\hat{s}$ and $\hat{t}$ before the width compaction. Then, these avoid that another box is shifted above a source or below a sink. Further, argument maps may contain cycles (although they rarely do). In a preprocessing step we remove cycles by reversing the direction of edges in G such that it becomes cycle-free. Since computing a cardinality minimal set of such edges is NP-complete [127], we use the Open Graph Drawing Framework (OGDF)[1] implementation of a well-known heuristic by Eades et al. [76] to solve this problem. In the last step, when we remove the nodes $\hat{s}$ and $\hat{t}$, we need to correct the direction of those edges we have reversed in the preprocessing step.

As already mentioned in Section 5.3, we randomize the computation of the topology, which we do using OGDF. We do this at two points: (i) for the construction of the feasible upward planar subgraph and (ii) we randomize the order in which the deleted edges are reinserted. Each of these two steps is performed ten times, i. e., we execute the computation of a feasible upward planar subgraph 100 times in total. Over all runs we take the crossing minimal upward planar representation as the result. We implemented the algorithms in C++. All experiments were performed on a single core of an Intel Xeon E5-2670 processor that is clocked at 2.66 GHz. The machine is running Linux 3.4.28-2.20 and has 64 GiB of RAM. We compiled our C++ implementation with GCC 4.7.1, using optimization level 3.

The basis of this evaluation is a set of 51 argument maps differing in size, purpose of usage and experience of the creator (uploaded to http://www.graph-archive.org). For a more detailed analysis of these input graphs we refer to [71]. First, we present statistics about measurable quality criteria, and then discuss the æsthetic qualities of these layouts.

STATISTICS. We start by analyzing the number of crossings in the final layout as well as in the upward planar representation $(\mathcal{U}, \Gamma)$. Although it is theoretically possible that the final layout contains fewer crossings than $(\mathcal{U}, \Gamma)$, for all instances there are at least as many crossing in the final layout as in the upward planar representation $(\mathcal{U}, \Gamma)$. However, if $(\mathcal{U}, \Gamma)$ is free of crossings, then the final layout is planar as well (compare to Lemma 5.1).

We note that, in our experiments, on average there are 69% more crossings in the final layouts than in the upward planar representations. Although this number may seem high, this is only because most of our input drawings have little or even no crossings. Then, even a slight increase in the number of crossings yields a high relative increase of crossings.

1 http://www.ogdf.net

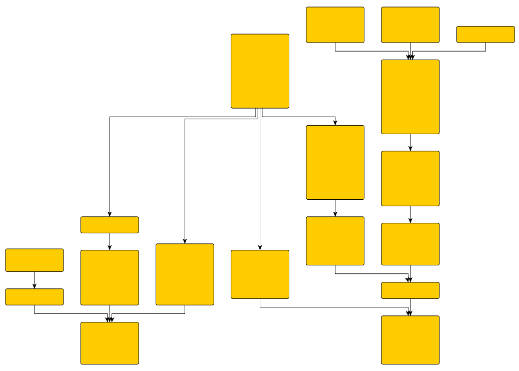


Figure 5.12: Instance D-2.5.

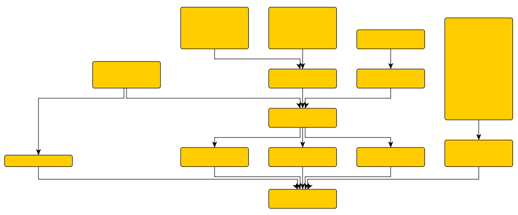


Figure 5.13: Instance D-5.3.

Due to the shape algorithm there can be at most four bends per edge. Except for three bend-free layouts, the average number of bends per edge is between 0.4 and 1.5. Only 13 of 51 layouts have edges with four bends. Taking the average over all layouts, we have 1.06 bends per edge.

Finally, we turn towards the runtime analysis. The maximum running time for the 51 input instances is 10.25s for an instance having 134 nodes and 158 edges, whereas the average is 0.32s. The average running time is strongly influenced by the few large instances. For instances with at most 30 nodes, the running time is 0.02s on average while the maximum runtime is 0.19s.

It is interesting to note the contribution of the three phases topology, shape and metrics to the overall runtime. Shape and metrics make only insignificant contributions of less than 0.01% and 0.2%, respectively, whereas the topology step needs 99.6% of the overall runtime. Thus, the topology step is the bottleneck.

We repeated our experiments with a decreased number of randomized runs in the topology step from 100 to 9. The maximum and average running time for the topology step decreased from 10.25s to 0.98s and from 0.32s to 0.03s, respectively. This decrease to roughly 1/10 in running time is not surprising as it simply reflects the fact that the topology step requires nearly 100% of the time for computing the layouts. Nevertheless, this is a significant improvement in running time while the average number of bends per edge increases only slightly from 1.05 to 1.07, and the average number of crossings increases from 2.64 to 2.76 for our instances.

case studies. Here, we present three layouts that are created by the algorithms described in Section 5.2–5.5. Using these layouts as examples, we discuss the benefits and the disadvantages of our algorithms. Figure 5.12 shows a layout for which we have no suggestions for improvement. The layout is free of crossings, the number of bends are close to the minimum, it is compact and locally symmetric. Beside the hard facts, it is well structured and gives a good impression of the internal structure. The other two layouts, while looking good in general, have minor drawbacks, which we discuss in the following.

In Figure 5.13 on the left side there is a configuration that is similar to a bow which we treated in Section 5.5. However, this time, we have a bow containing a box. We could expand our concept to paths in $\widehat{G}$ that contain only nodes whose in-degree and out-degree equals one. Such a scenario is not treated in our algorithm. Since there can be many different special cases which might be resolved easily, we propose to investigate and integrate additional algorithms for optimizing orthogonal drawings into our approach.

In Figure 5.14 there is a set of boxes on the right side of the layout that is positioned quite high. It seems as it might be possible to move these boxes downwards. However, these boxes are aligned with boxes that are some columns apart, i. e., they need to be

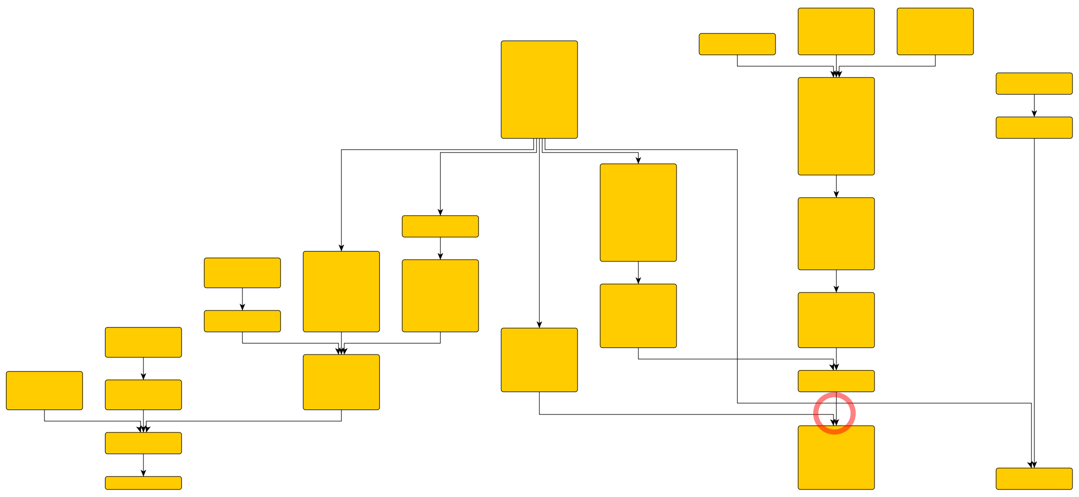


Figure 5.14: Instance D-2.7. Circle added to highlight the effect of different edge-edge spacings.

positioned that high. One could reconsider the drawing style, such that alignment of boxes is only required if the number of columns between the boxes is small. Note that we can easily integrate such a modification by adapting the computation of the groups in Section 5.5. Besides this drawback, the example gives a good impression of the edge bundling enabled by the minimum edge-edge spacings. The effect the two different edge-edge spacing constraints have on the final layout can be seen inside the red circle. The two edges that have the same target have a small distance between them while the third horizontal edge-segment belonging to the third edge is placed at a greater distance to the other horizontal edge-segment.

Overall, we conclude that the computed layouts are of high quality from an æsthetic point of view as well as regarding the objective quality measures.

5.7 CONCLUSION

Our main result is an algorithmic approach to generate column-based graph layouts for directed acyclic graphs. Our approach integrates the layer-free upward crossing minimization technique by Chimani et al. [55] into the well-known topology-shape-metrics framework. We showed that several of the arising subproblems are NP-complete and devised efficient algorithms, many with provable quality guarantees, for all steps. In the first step we minimize the number of crossings as well as the total source/sink distance using layer-free upward crossing minimization. Afterwards, we use a modified graph drawing heuristic by Biedl and Kant [28] such that there are at most four bends per edge. Thereby, nodes and edges are assigned to columns. In the last step, we first minimize vertical edge length using a greedy approach and then horizontal edge lengths based on iteratively removing rightmost compaction paths. During the edge length minimization we remain true to the columns such that the resulting layouts are column based.

Unconventionally, we do not fix the topology computed in the first step throughout the remaining phases. When computing the column assignment in the second step, we relax some of the decisions we made before, and hence, the focus of the algorithm is shifted away from minimizing the number of edge crossings. We fix the edge crossings again when minimizing the vertical edge length. In usual applications of the topology-shape-metrics framework crossing minimization is the first aspect that is considered and all remaining optimization criteria are treated afterwards and therefore must handle the choices made in the first step. This is not the case for our approach, where we increase the significance of bend and total edge length minimization.

The layouts we compute are of high quality. They have a clear, well-structured look and are compact. Input instances that have a typical size for argument maps can be layouted quickly. Thus, our algorithm is feasible for practical purposes. In fact, the algorithm has, in the meantime, been integrated into the tool argunet[2], which is used for creating and manipulating argument maps, and is now available as a plugin.

Although the generated layouts are of high quality, certain unappealing features remain and as a next step it may be worthwhile to explore if it is possible to adapt algorithms to optimize the layout of orthogonal drawings to our drawings; e. g., the 4M algorithm [88]. Coming from the practical application another interesting step is to investigate how to handle dynamic argument maps, i. e., generate drawings of argument maps that change over time while maintaining the user's mental map.

2 `http://www.argunet.org`, argunet is open source and published under the GPL

Part II

LABELING MAPS AND DIAGRAMS

6

INTRODUCTION

In this chapter we give a general introduction to the problem of map labeling. We begin by discussing the beginnings of map labeling in the algorithmic community and, then, in the then following section we proceed to give a more detailed description of the main algorithmic results. After we have covered this, we turn towards the problem of labeling dynamic maps. We call a map *dynamic* if it *allow* continuous dynamic operations, e. g., zooming. Due to the increased popularity of hand held devices such as smart phones there are now many applications which support dynamic maps (e. g., the vector based Google Maps for Android). There have been significantly fewer algorithmic results that deal with labeling dynamic maps. After we have discussed results dealing with dynamic map labeling we look briefly into the problem of *boundary labeling*. In this problem setting the goal is still to place text, but not *on* the map, but on the boundary of it. The feature we want to label is connected to the label by a *leader*. A leader is usually a combination of multiple straight-line segments. Although boundary labeling certainly has applications in the field of labeling maps, more often can this style be observed for labeling diagrams (e. g., medical illustrations), or panoramas. Finally, in the last section of this chapter, we give a detailed description of the contribution in the following three chapters of this thesis.

One of the many important aspects of creating maps is the *placement* of labels, be it text (e. g., names of towns or cities), or symbols (e. g., highway shields). This part of the map making process is usually referred to as the "placement of names on the maps" or as the "labeling of the map". Although at first glance this might seem to be a not so difficult task it still requires more than 50% of the time necessary for the whole creation of a map [69]. Not only is it time consuming, but it is also of tremendous importance. As the cartographer Imhof put it "Good name position aids map reading considerably and enhances the esthetics of the map. [...] Poor, sloppy, amateurish type placement is irresponsible; it spoils even the best image and impedes reading." [123]. The Computational Geometry Impact Task Force determined that map labeling is one of the import areas of research in Discrete Computational Geometry [53], and hence, a considerable amount of work has been done to automate the process of placing text on a map. Besides the attention from the algorithmic community there are several professional software products from the field of geographic information science (GIS) that support map labeling. The software company esri has published MAPLEX [83] a commercial ArcGis

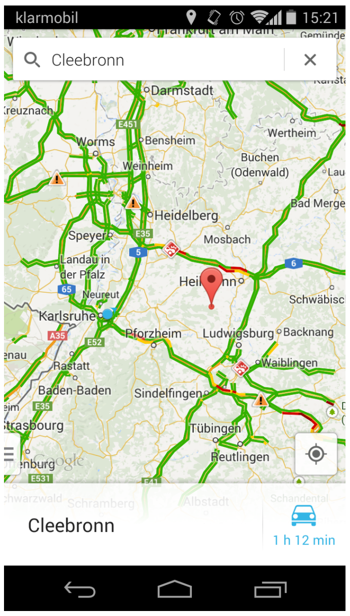


Figure 6.1: Google Maps. The label for one of Germany's largest cities *Stuttgart* is missing. Screenshot: ©Fabian Fuchs.

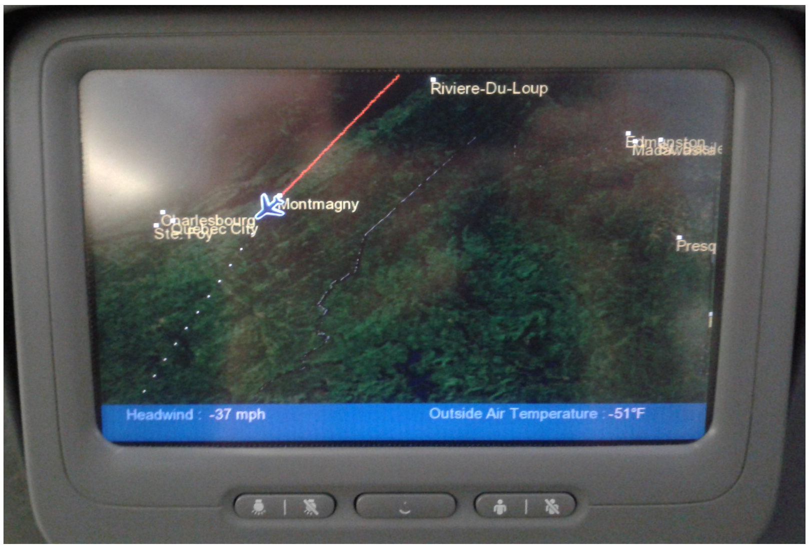


Figure 6.2: Picture of the on-board entertainment systems during a United Airlines flight. The dynamic map labeling algorithm exhibits multiple label overlaps.

extension that supports automatic map labeling. But there are also open source alternatives such as Mapertive [40] to do this. However, despite the progress in the automatization of map labeling there are still many unresolved questions concerning generating map labelings of high quality. This is especially true for the field of dynamic map labeling; see Figures 6.1 and 6.2 for two examples.

6.1 PRELIMINARIES

Still today one of the most highly regarded publications concerned with the question of how to place text on a geographic map is "Die Anordnung der Namen in der Karte" [122] by Imhof. This publication was later translated from German into English and subsequently published in 1975 under the name "Positioning Names on Maps" [123]. In this work Imhof specifies several basic rules that cartographers should follow when designing the labeling of a map. Among those rules are:

- *Legibility*. "The names should [...] be easily read, easily discriminated, and quickly located."
- *Clear graphic association*. "The name and the object to which it belongs should be easily recognized."

Further, he distinguishes between three types of map elements which can be labeled.

1. "*Position designations*, or names for point-like objects and concepts."
2. "*Linear designations*, i. e., names for objects [...] with linear [...] extent."
3. "*Areal or surface designations*"

One of the first publications in which map labeling was considered from an algorithmic point of view (i. e., in the sense that the labeling is considered to be a process which can be done automatically) was published in 1972 by Yoeli [188]. In it the author discusses "the logic of automated map lettering", i. e., the author gives a general overview of an approach to automatically label a map. He discusses the necessary prerequisites (e. g., aquiring data, deciding what to label and in which font) for this problem, the challenges and he gives a

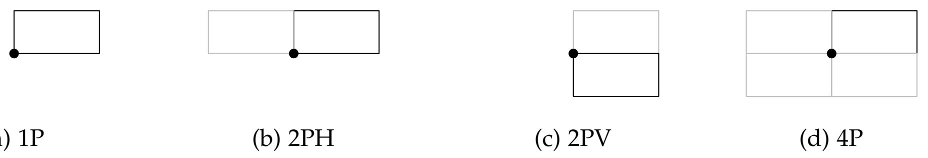


Figure 6.3: Depicted are the most common variants of the fixed-position model.

brief description of an algorithm to label a map (although the description is lacking some depth). Like Imhof, he distinguishes between the same types of map elements but denotes them differently:

1. "Names of points or items which cover a small area."
2. "Names of lines or bands (e. g., rivers, boundaries, etc.)."
3. "Names of areas."

Yoeli also describes the general process of lettering a map in three steps:

a. "The choice of the name contents and its classification (settlements, rivers, etc.)."
b. "Determination of type characters and sizes"
c. "Placement."

In the algorithmic community the first two steps are usually omitted, and it is assumed that the labels, i. e., the text and font type, are part of the input and are not subject to optimization. Further, most research in the algorithmic community is concerned with labeling "Position Designations"/"Names of points or items which cover a small area.". The problem is most often called the POINT-FEATURE LABELING PROBLEM (PFLP). In the following section we give a broader overview of the PFLP, its variations and well-known results, relevant to this thesis.

THE POINT-FEATURE LABELING PROBLEM (PFLP). In the following we discuss the point-feature labeling problem as it is the most common variant of the map labeling problem investigated in the algorithmic community.

For the PFLP we are given a set of points (also called *anchors*), and a corresponding set of labels (usually rectangles) and the goal is to place the labels on the map. The allowed positions of the labels on the map (i. e., in the two-dimensional plane) are restricted. The most common models used (in the algorithmic community) are the fixed-position model, and the slider model.

fixed position model

In the *fixed position model* there are for each point-feature certain fixed positions in the plane the label can be placed at. Usually, the positions are such that a corner of the label coincides with its own anchor. In the 1P model the label's lower left corner is placed on its anchor point. There are two different 2P models, 2PV and 2PH. In the 2PV model the label must be positioned such that either its lower-, or its upper-left corner coincides with its anchor. Finally, the 4P model requires that one of the label's corner is positioned on the label's anchor. For an illustration of these models see Figure 6.3. Although the mentioned fixed-position models are the most commonly used, there are other variants of the fixed-position model that have been considered.

slider model

In the *slider model*, which was originally proposed by van Kreveld et al. [136], there are *no* fixed positions for the labels, but instead of requiring that a certain corner of the label coincides with the anchor, it is required that the label is positioned such that the anchor lies on the border of the label. One might argue that the slider model is based on the fixed-position model in the sense that the fixed positions are the "extreme" positions for

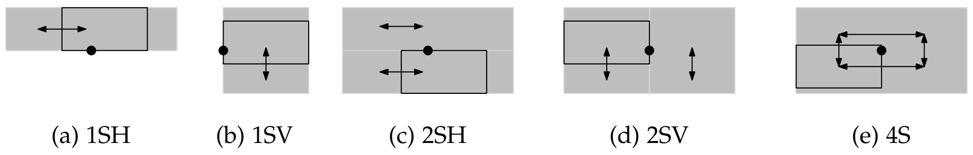


Figure 6.4: Depicted are the most common variants of the slider model.

labels in the slider model. In the 1SV (1SH) model the label's anchor must lie on its lower (left) boundary, respectively. In the 2SV model the label's anchor must lie on the lower or upper boundary of the label. Similarly, in the 2SH model, the label's anchor must lie on the left or right boundary of the label. Finally, in the 4S model requires the label's anchor be on one of the four boundaries of the label; see Figure 6.4 for an illustration of the aforementioned models.

In a recent publication Reimer and Rylov [161] propose a multi-criteria model for the PFLP which summarizes rules and constraints for label placement that can be found in the literature. The constraints (slightly reformulated) by Reimer and Rylov are

- C1 Names must not overlap point feature symbols.
- C2 No two names may overlap.
- C3 No two point features may overlap.

The authors note that most research has been focused on these constraints. We note that many of the given rules (see below) concern the actual content of the map (e. g., coastal lines) which we did not consider in this work. Therefore, we omit quoting the rules at length but focus on the ones we feel are important for the problems considered in this thesis.

- R1 Type arrangement should reflect the classification, importance and hierarchy of objects.
- R2 Labels should be placed horizontally.
- R3 The lettering to the right and slightly above the symbol should be prioritized.
- R6 Names should not be too close to each other.
- R7 Labels should not be excessively clustered nor evenly spread out.
- R8 Each label should be easily identified with its point feature. Ambiguous relationships between symbols and their names must be avoided.

The authors assert that rule R6 and R8 are the most important ones. We will come back to this classification at the beginning of Chapter 7.

Now we are ready to give a brief overview of the known algorithmic approaches to tackle the map labeling problem. We will not give a detailed description of every publication on this topic, but we discuss those publications, we feel, are important to the results in this thesis. For a list of publications concerned with map labeling we refer the reader to the Map Labeling Bibliography [186] was maintained until 2009 by Alexander Wolff and Tycho Strijk. Please note that we distinguish between static maps and dynamic maps in this thesis and thus we use the qualifier static or dynamic for maps where applicable. Hence, the map labeling problem considered by most researchers is here the *static* map labeling problem.

6.2 STATIC MAP LABELING

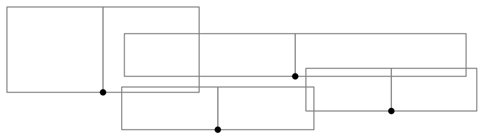
There are mostly three variants of the POINT-FEATURE LABELING PROBLEM for labeling static maps. The input to the problem consists of a set P of n anchor points in the plane, and a set L of axis-aligned, rectangular labels where one label in L corresponds to one anchor in P. *valid labeling* The problem then asks for a *valid labeling* $\mathcal{L}\colon L' \subset L \to \mathbb{R}^2$ that assigns each label in the set $L' \subseteq L$ a position in $\mathbb{R}^2$ which is permissible in the model used (fixed-position or slider model). The decision problem is (usually) asking whether a labeling for all labels in L exists, such that no two labels overlap (and hence requires $L' = L$. Unfortunately, this problem is NP-complete [141] even for simple models (but not for the 1P fixed-position model; the decision problem is in P). There have been several optimization problems proposed which are based on the PFLP. Those problems consists of two parts, that, unfortunately cannot be considered separately. One part is to determine the set $L' \subseteq L$ of labels that belong to the optimal solution, while the other part consists of determining the position of those labels contained in L'. We discuss in the following the most common optimization problem for PFLP in static map labeling.

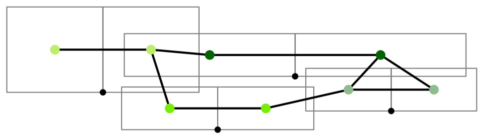
6.2.1 *The (Weighted) Label Number Maximization (LNM)*

One of the standard problem in algorithmic map labeling is the LABEL NUMBER MAXIMIZATION (LNM) problem. This problem is a selection and placement problem. The optimization goal is to determine a labeling where the set $L' \subseteq L$ is maximum among all valid labelings where no two labels overlap. Note that a labeling where two labels *do* overlap may violate the requirement put forth by Imhof that the labeling should be *legible* and *clear graphic association* should be possible; see Figure 6.2 for a real-word example where the labeling is not legible because of overlapping labels. Unfortunately, the label number maximization problem is NP-hard, even in the 1P fixed-position model, and even when the labels are unit-squares already [121, 89].

We discuss first algorithmic results for this problem in the fixed-position model, and afterwards turn towards results in the slider model. At the core of LNM in the fixed-position model lies a maximum independent set problem [1]. To see this, place for each anchor copies of its label on all its permissible positions, and compute the intersection graph I of this set of rectangles. Since all rectangles that share the same anchor have (at least) this point in common, at most one label per anchor can be selected. Now, an optimal solution of an LNM instance corresponds to a MAXIMUM INDEPENDENT SET on I. For an illustration of this concept see the example depicted in Figure 6.5. Since the actual text of the labels is, from an algorithmic point of view, irrelevant the problem has been called the MAXIMUM INDEPENDENT SET OF RECTANGLES (MISR) problem. The natural extension to weighted rectangles has been considered also which is known as the MAXIMUM WEIGHTED INDEPENDENT SET OF RECTANGLES (MWISR) problem.

While it is known that approximating MAXIMUM INDEPENDENT SET within $n^{1-\varepsilon}$ on general graphs is NP-hard [112], for the restricted graph class of rectangle intersection graphs better results are known. Agarwal et al. [1, 2] proposed an $O(n \log n)$ time and $O(\log n)$ factor approximation algorithm. Berman et al. [23] provided a $\lceil \log_k n \rceil$-approximation with time complexity $O(n^{k+1})$ for any integer $k \geq 2$, upon which Chan [50] improves by proposing two algorithms with time complexities $O(n \log n + n\Delta^{k-2})$ and $n^{O(k/\log k)}$, respectively. In 2009 Chalermsook and Chuzhoy [49] published a paper taking the next step towards a constant-factor approximation for MISR. In their publication, the authors provide a $O(\log \log n)$ approximation algorithm. Recently, Adamaszek and Wiese have shown that there exists a QPTAS with quasi polynomial running time of $O(2^{\mathrm{poly}(\log n/\varepsilon)})$ for MWISR.


(a) 2PH instance.


(b) Intersection graph.

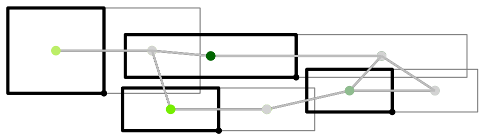

(c) Optimal solution. The chosen label positions marked in bold black lines, while the chosen vertices of the MIS instance are not grayed out.

Figure 6.5: A 2PH instance is depicted, along with its intersection graph and the optimal solution to the LNM problem and the corresponding MIS of the intersection graph.

At the time this thesis was written, it is still unknown whether there exists a PTAS or a constant factor approximation algorithm for M(W)ISR.

For map labeling there can be sensible restrictions on the shape of the rectangles used, which then can be exploited to obtain better approximation algorithms. Often it is assumed that the labels all have the same height, which is not unreasonable since the labels might depict text with the same font size. We refer to the problem of finding a maximum independent set of unit-height rectangles as UNIT-HEIGHT MISR (UHMISR)). There is also the case that the labels are not used to depict text, but rather to show pictograms, which can be assumed to be squares. Then, it is not unreasonable to assume that the dimensions of all labels is the same. Similar to the UHMISR we refer to the problem of finding a maximum independent set of unit-squares as UNIT-SQUARE MISR (USMISR). Note that if the all rectangles have the same dimensions, but are not squares, we can transform the entire instance by stretching the plane appropriately, such that the rectangles become squares. Both problems, the UHMISR and USMISR, can also be adapted for the weighted case.

In contrast to the general MISR problem, the existence of a PTAS is for the UHMISR problem is well-known. Agarwal et al. [2] showed this by describing a $(1 + 1/k)$-approximation algorithm with time complexity $O(n \log n + n^{2k-1})$, for any $k \geq 1$. The basis for their PTAS is the so-called *shifting* technique proposed by Hochbaum and Maass [117]. The PTAS proposed by Agarwal et al. [2, 1] has been extended such that it can also handle weighted labels [160] in $O(n \log n + n^{2k-1})$ time. In 2004 Chan gave a PTAS with a time complexity in $O(n \log n + n\Delta^{k-1})$ for the unweighted case [50], where Δ is the maximum degree of the intersection graph.

We now turn towards the algorithmic results in for the POINT FEATURE-LABELING PROBLEM in the slider-model. In the publication in which the slider model was introduced, the authors already provided an NP-hardness proof for the LNM in the 4-slider model even if all labels are unit squares. Furthermore, they gave approximation algorithms for the different models under the assumption that all labels have the same height, but may have label-specific widths. However, the authors did not provide any solution for the case that the labels have a weight and the goal is to maximize the total weight of all displayed labels. This problem was considered by Poon et al. [160]. The authors analyze bounds for the ratios between optimal solutions in the nine different models (fixed-position and slider). They show that label number maximization problem for unit-height rectangles in the slider model is NP-hard even in the one dimensional case. However, their main contribution is a

$(1/2 - \varepsilon)$-approximation algorithm for the weighted label number maximization problem for unit-height rectangles in the slider model. The algorithm's time complexity is $O(n^2/\varepsilon)$ and its space complexity is $O(n/\varepsilon)$. The existence of a PTAS for the weighted 1SH model for unit-height rectangles was one of the open questions posed in this paper. With a novel approach for constructing polynomial-time approximation schemes Erlebach et al. [82] were able to answer the open question positively. The authors give PTASes for various models for the UHMSR problem and even for the variant where the ratio between the maximum and minimum height of the rectangles is bounded by some constant $c \geq 1$.

Besides the numerous approximation algorithms for the LABEL SIZE MAXIMIZATION problem in various models there have also been many attempts at finding algorithm that perform well in practice, but for which no quality guarantee can be made. One way to tackle the problem is by using meta-heuristics, which has been done quite extensively [116, 68, 187, 5], and by heuristics which are specifically developed for this problem [70, 164]. For an overview of performance of many different LNM algorithms we refer the reader to [59].

6.2.2 *Other Point-Feature Labeling Problems*

Although maximizing the number of displayed labels seems to be the most natural optimization problem for static map labeling, other variants of the PFLP have been considered. We mention some of them here briefly along with important results.

Formann and Wagner [86] introduced the LABEL SIZE MAXIMIZATION PROBLEM (LSM) that asks for the largest stretch factor ϕ by which all labels can be stretched such that no two labels intersect. The authors prove NP-hardness for the 4P fixed-position model, and a polynomial time algorithm for the 2P fixed-position model based on 2SAT (recall that 2SAT is in P [137]). On the basis of this polynomial-time algorithm Formann and Wagner give an optimal 1/2-approximation algorithm for the 4P fixed-position model. The algorithm is optimal with respect to its time complexity of $O(n \log n)$ and to its approximation ratio (unless P = NP). In the paper [136] in which the slider-model was introduced, an NP-hardness proof for the LSM in the slider-model was provided, along with PTASes and simple 1/2-approximation algorithms. As with the LNM problem, there have also been publications that explore the quality performance of meta-heuristics [78]. For an experimental evaluation of several label size maximization algorithms see [57].

Another problem where all points must be labeled, but the labels sizes cannot be altered, is the LABEL OVERLAP MINIMIZATION (LOM) problem that asks for minimizing the number of label overlaps [130, 58, 162]. Motivated by the problem of displaying planes with additional information (e. g., height, velocity) on a radar screen Gerrits et al. [20] investigated the FREE-LABEL MAXIMIZATION (FLM) problem. The FLM problem has the same requirement as the LOM but the goal is to maximize the number of non-overlapped labels. The authors show NP-hardness and give approximation algorithms for all the nine labeling models. In another publication the authors give a heuristic and evaluate its performance experimentally [21].

6.3 DYNAMIC MAP LABELING

While the static map labeling has been investigated quite thoroughly, as we have seen in the last section of this chapter, there has been comparatively little work regarding dynamic map labeling, an area which has become more important with the increased popularity of mobile navigation devices, smart phones, or desktop applications for maps (e. g., Google Maps, or Google Earth).

In this thesis we say a map is *dynamic* if it supports one or more of the following three continuous operations:

1. zooming in/out

2. rotating
3. panning

Since dynamic maps are characterized by continuous operations (e. g., panning) a certain sense of consistency in the visual depiction of the map is desirable. Although there is some literature regarding concepts called *frame coherency* [4] or *temporal continuity* [18], Been et al. [9] were the first to introduce the concept of *consistency* for displaying dynamic maps. They identified four important concepts for consistency: "(D1) Except for sliding in or out of the view area, labels should not vanish when zooming in or appear when zooming out. This captures the usual expectation that strictly more features get labeled as we zoom in, and strictly fewer features get labeled as we zoom out. It ensures that labels do not appear, disappear and then re-appear under monotonic zooming. [...] In some cases we prefer a less restricted version of this desideratum, where we might want, for example, a country label to disappear as we zoom in to street level detail. (D2) As long as a label is visible, its position and size should change continuously under the pan and zoom operations. (D3) Except for sliding in or out of the view area, labels should not vanish or appear during panning. If (D2) is satisfied, then (D3) can be satisfied by making the decision to select a label a function of scale. (D4) The placement and selection of any label is a function of the current map state (scale and view area)." [9]. The authors further conclude that the displayed view is only a function of the scale and thus does and should not depend on the history of the operations used before. We note here that the authors focused their analysis only on the operations panning and zooming. However, it is straightforward to adapt this to rotating.

Since maps are solely generated for humans, it would be interesting to investigate how dynamic maps should be designed so that they benefit users the most. Although there are several studies that are concerned with interactive maps [115, 135, 149], to the best of our knowledge there is only one publication [154] which considers the problem of how to design dynamic maps for humans. However, this paper focuses on panning which we do not discuss in this thesis.

Besides the question of what a consistent labeling actually is, the optimization function is not obvious. Been et al. [10] gave an in-depth analysis of the dynamic map labeling problem for the zooming operation. They consider only a 1P fixed-position model. In their paper, the authors define an *active range*, that is the scales for which the label is visible. Due to their consistency requirement each label has at most one contiguous active range. The optimization function is then to maximize the total length of all active ranges of an instance. Note that this is very similar to the LABEL NUMBER MAXIMIZATION problem in that it maximizes the number of visible labels over all scales. This problem is, unsurprisingly, NP-hard, but the authors give approximation algorithms.

The problem of labeling zooming maps has been considered before by Poon and Shin [159]. They propose an algorithm that precomputes solutions for a number of scales and interpolates between them. However, the solutions for the scales are obtained independently and thus there is no guarantee for consistency (at least in the sense proposed by Been et al. [9]). Petzold et al. [158, 156] also use a preprocessing step and apply this approach to labeling of dynamic maps that allow panning. Mote [157] discuss the same problem but proposes an algorithm that omits a preprocessing step completely. However, both approaches cannot guarantee that the solution is consistent.

The problem of labeling rotating maps was first discussed by us in a previous publication [104] which contains the preliminary results that are the basis for Chapter 7. Since then there have been two new publications which also look into the problem of labeling rotating maps. Yokosuka and Imai [189] consider the problem of label size maximization for the 1P fixed-position model for rotating maps. They show that the problem can be solved efficiently in this model (as in the case for static maps). An interesting open question would be to consider different fixed-positions models, or even the slider model and apply it to this problem. Gemsa et al. [100] looked into the problem of labeling dynamic maps for small screen devices such as GPS navigation devices. These usually allow the three operations panning, rotating, and zooming. Besides an NP-hardness proof and approximation algorithms they look into the problem of displaying not too many labels at the same time which is an important feature for small screen devices. Further, they describe a very general labeling model which allows to be adapted to other variants of dynamic labeling.

At the end of this section we want to point out that the term dynamic in the context of labeling applies to many different scenarios and other authors have used it for other meanings [6, 84]. Recently, Buchin and Gerrits have, motivated by air traffic controlling, investigated the case of dynamic point labeling. In this context the word 'dynamic' means that the points are allowed to move. As it turns out this problem is even in simple cases PSPACE-complete [44]. However, de Berg and Gerrits proposed a heuristic for this problem that performs well in practice [21].

6.4 BOUNDARY LABELING

In this section we give a brief discussion of a different concept of labeling called *boundary labeling* which can be seen as a variant of static map labeling. The reader might wonder why we chose to separate this problem from Section 6.2 in which we discussed several static map labeling problems. Although this problem is in the literature often motivated by map labeling, we feel that this problem is much more suited to labeling diagrams (e. g., medical diagrams), and thus should belong into a distinct category. To conform to the established terminology, we will however, also say we label a *map*.

The labeling models we covered in the preceding sections required that all labels are placed such that their boundary touches the point feature it is supposed to label. This requirement, however, can have detrimental effects on the visual appearance if, for example, we are dealing with dense point sets, or if the labels are large compared to the map/diagram (which is a typical problem encountered when illustrating diagrams). To overcome this problem Freeman et al. [108], and Zoraster [190] suggested to place labels not directly at the anchor but at sensible places removed from the actual anchor and then connect each label with its corresponding anchor with a line (called *leader*). This idea was taken up on by Bekos et al. in their paper "Boundary labeling: Models and efficient algorithms for rectangular maps" [15] in which the authors introduced *boundary labeling*.

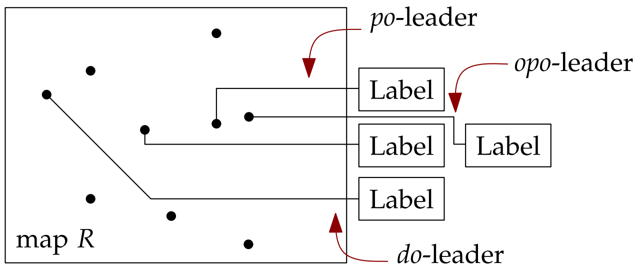


Figure 6.6: Different types of leaders for boundary labeling.

For boundary labeling it is assumed that the input consists of an axis-aligned, rectangular map M, and a set P of points inside the map. Further, for each point in P there is a corresponding rectangular label given. The goal is then to find positions for the labels that are placed outside of M such that no two labels intersect, and then to connect the points with their corresponding labels without having intersections between the leaders. The boundary labeling models differ in three aspects: i) the type of leader, ii) the sides of the map where labels may be placed, and iii) whether there are fixed positions for the labels (and the points where the labels connect with the leaders), or if the labels can be positioned freely.

The different types of leaders are identified by a string consisting of one ore more of the following letters: p – parallel, o – orthogonal, d – diagonal, s – straight. A *po*-leader,

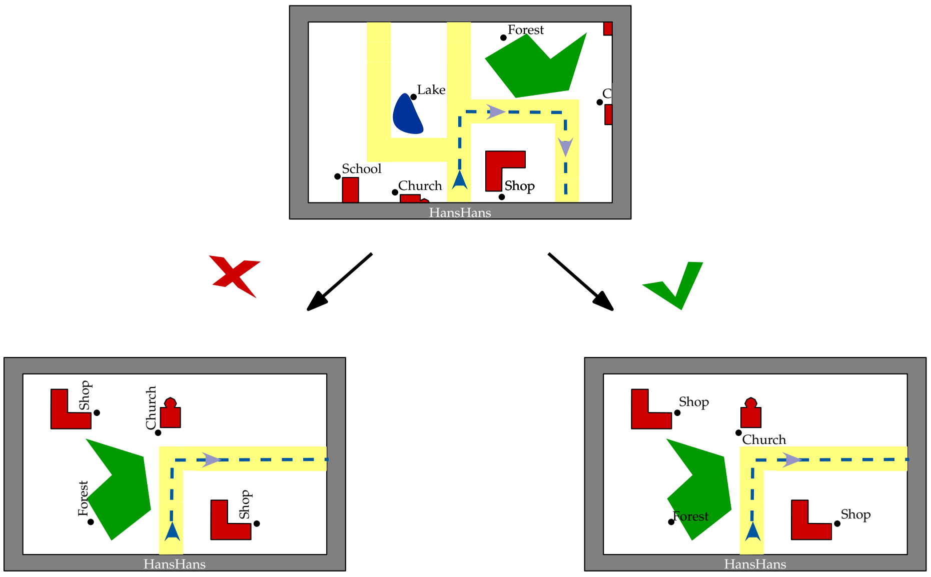


Figure 6.7: GPS navigation devices usually depict the map in such way that the driver always drives "up". Hence, the device needs to rotate the map.

for example, is a leader that consists of a parallel line (parallel with respect to the side of the map the leader connects to), and a subsequent second line that is orthogonal (to the side of the map the leader connects to); for an example with different types of leaders see Figure 6.6.

Contrary to the previously discussed static map labeling problems, many problems in the boundary model can be solved efficiently, i. e., they can be solved by polynomial-time algorithms. Often, but not always, those algorithms are dynamic programming algorithms and exploit that the solution must not contain crossing leaders. This means that when connecting a point with a label by a leader this, when choosing the leader in the "right way", separates the input instance into two independent subinstances—which is usually a sign that a dynamic programming approach might work.

For a comprehensive introduction for boundary labeling algorithms, as well as a detailed description of the model and its application we refer the reader to the journal paper by Bekos et al. [15].

6.5 OVERVIEW OF THE FOLLOWING CHAPTERS

We have now laid the basis for the following chapters. Here, we give brief summary of what we discuss there.

In the following chapter, we discuss a dynamic map labeling problem that arises in the context of modern mobile devices, such as GPS car navigation systems; see Figure 6.7. But it also occurs with applications for modern desktops that support dynamic maps, e. g., Google Earth. As we have stated before there are three basic operations that dynamic maps may allow. For this chapter we discuss the problem of labeling maps that can be continuously *rotated*. We begin by proposing a model for labeling rotating maps, which is based on the consistency requirements proposed by Been et al. [9] for labeling dynamic maps. We investigate complexity (and show NP-hardness), develop exact ILP formulations, efficient approximation algorithms, as well as simple but fast heuristics. The chapter is concluded by an evaluation of i) the model, and ii) the proposed algorithmic approaches on real-world data. To the best of our knowledge, this research was the first to consider this problem from a theoretical computer science point of view.

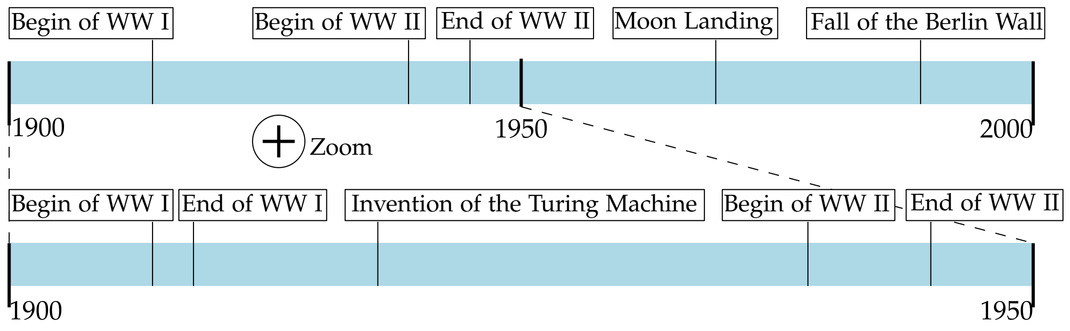
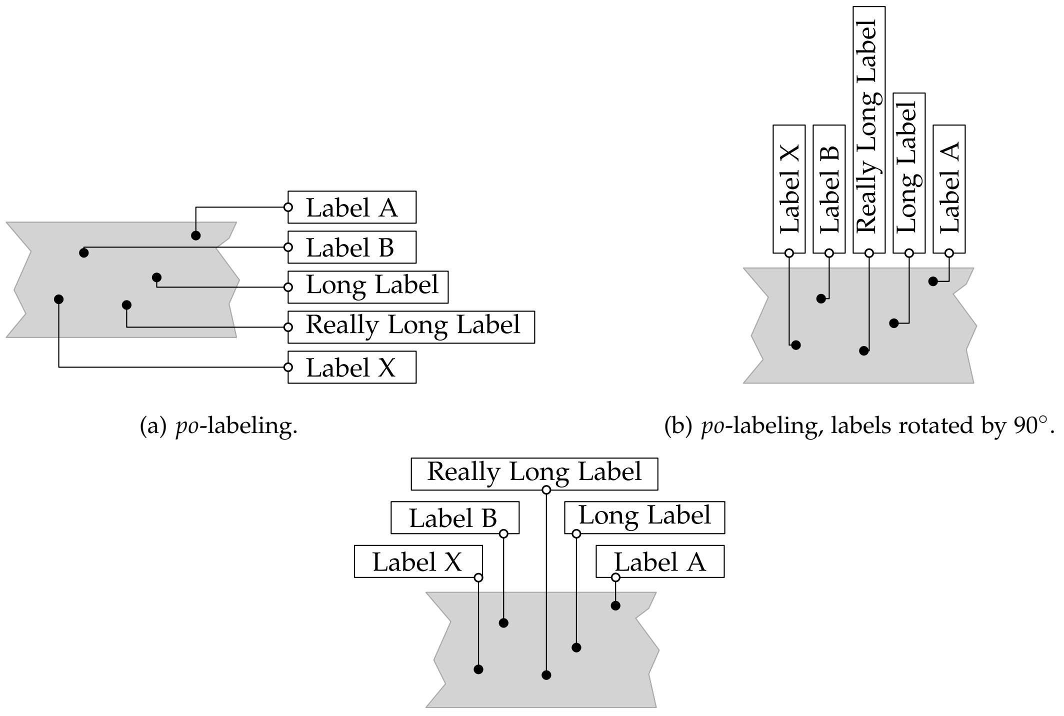


Figure 6.8: Timeline of important historical events in the 20th century.



(a) *po*-labeling.

(b) *po*-labeling, labels rotated by 90°.

(c) Labeling in our model.

Figure 6.9: Labeling of a panorama with different approaches.

After we have discussed labeling of rotating maps, we turn, in Chapter 8, towards the second basic operations allowed for dynamic maps: *zooming*. Contrary to the problem of labeling maps which allow rotation, labeling maps that allow zooming has already been considered, e. g., by Been et al. [10]. However, previous publications in the field of theoretical computer science only considered the fixed-position model. We improve on this by investigating the dynamic map labeling problem for zooming maps in the slider model. We begin by investigating the labeling of one-dimensional dynamic maps, i. e., lines. Although at first glance this seems only like a purely academic problem with no real application value, we see applications, e. g., in the labeling of time lines. In Figure 6.8 we present a timeline of important historical events in the 20th century. Since not all events can be shown (there are simply way too many) "zooming" into the timeline (i. e., looking at decades/years more closely) frees up space to show more events. We extend our results for one-dimensional maps to the two-dimensional case and discuss the algorithms with a brief experimental evaluation.

In the last chapter of this part of the thesis we discuss the problem of labeling panoramas in the boundary model. Panoramas distinguish themselves from the usual objects of illustration by their property that they are unusually wide; see Figure 6.10. Most boundary labeling approaches position the labels to the right of the map; see Figure 6.9(a). However, for panoramas this quickly leads to visually unpleasing layouts, since the labels cannot exploit the large width of the panorama but need to be stacked at the right border. To deal with this problem one might propose to simply rotate the labels by 90° and position the labels at the top; see Figure 6.9(b). While such a layout can exploit the width of the panorama it might lead to non-compact drawings, since labels might be very long. We propose a different model, in which the labels are placed either above or below the panorama 6.9(c). As common for boundary labeling models the anchors and their labels are connected by a leader, however, contrary to most algorithmic approaches for boundary labeling, we restrict the type of the leaders to straight lines. More specifically, we allow only vertical lines as leaders. In this chapter we propose several algorithms (most of them are polynomial-time algorithms, but some investigated problems are NP-hard, and we either provide an ILP formulation, or when possible, a pseudo-polynomial time algorithm) and investigate their performance.

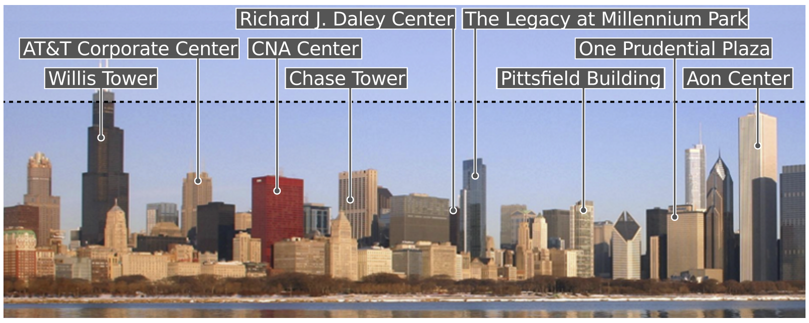


Figure 6.10: Panorama labeling for the skyline of Chicago. Photography: ©J. Crocker.

7

LABELING OF ROTATING MAPS

Dynamic maps that allow continuous map rotations, e.g., on mobile devices, encounter new issues unseen in static map labeling before. We study the following dynamic map labeling problem: The input is a static, labeled map, i.e., a set P of points in the plane with attached non-overlapping horizontal rectangular labels. The goal is to find a *consistent* labeling of P under rotation that maximizes the number of visible labels for all rotation angles such that the labels remain horizontal while the map is rotated. A labeling is consistent if a single *active* interval of angles is selected for each label such that labels neither intersect each other nor occlude points in P at any rotation angle.

We first introduce a general model for labeling rotating maps and derive basic geometric properties of consistent solutions. We show NP-hardness of the active interval maximization problem even for unit-square labels. We then present a constant-factor approximation for this problem based on line stabbing, and refine it further into an efficient polynomial-time approximation scheme (EPTAS). Further, we extend the EPTAS to the more general setting of rectangular labels of bounded size and aspect ratio. For the case that we want to compute solutions quickly (e. g., real-time application on mobile devices) we propose several heuristics.

This chapter is based on joint work with Martin Nöllenburg, and Ignaz Rutter. Preliminary results have been published in [103, 104, 106].

7.1 INTRODUCTION

Dynamic maps, in which the user can navigate continuously through space, are becoming increasingly important in scientific and commercial GIS applications as well as in personal mapping applications. In particular GPS-equipped mobile devices offer various new possibilities for interactive, location-aware maps. A common principle in dynamic maps is that users can pan, rotate, and zoom the map view. Despite the popularity of several commercial and free applications, relatively little attention has been paid to provably good labeling algorithms for dynamic maps.

Been et al. [9] identified a set of consistency desiderata for dynamic map labeling. Labels should neither "jump" (suddenly change position or size) nor "pop" (appear and disappear more than once) during monotonous map navigation; moreover, the labeling should be a function of the selected map viewport and not depend on the user's navigation history. Previous work on the topic has focused solely on supporting zooming and/or panning of the map [9, 151, 10], whereas consistent labeling under map rotations has not been considered prior to this work.

Most maps come with a natural orientation (usually the northern direction facing upward), but applications such as car or pedestrian navigation often rotate the map view dynamically to be always forward facing [107]. Still, the labels must remain horizontally aligned for best readability regardless of the actual rotation angle of the map. A basic requirement in static and dynamic label placement is that labels are pairwise disjoint, i.e., in general not all labels can be placed simultaneously. For labeling point features, it is further required that

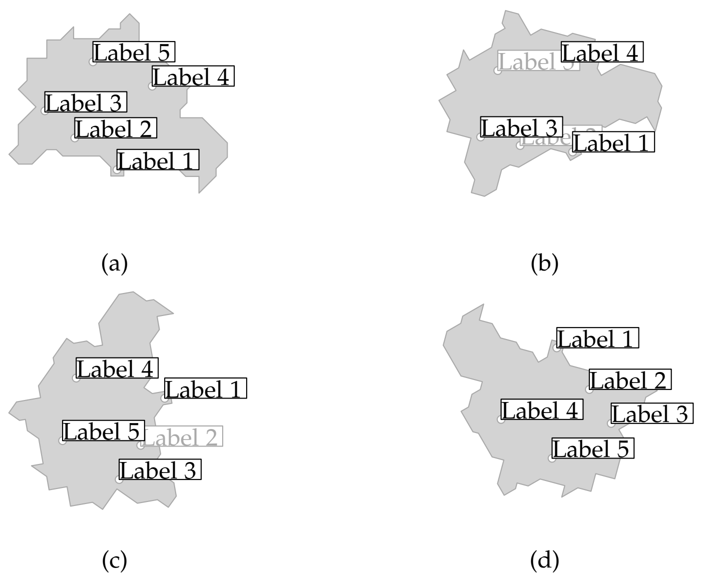


Figure 7.1: Input map with five points (a) and three rotated views with some partially occluded labels (b)–(d).

each label, usually modeled as a rectangle, touches the labeled point on its boundary. It is often not allowed that labels occlude the input point of another label. Figure 7.1 shows an example of a map that is rotated and labeled. The objective in map labeling is usually to place as many labels as possible. Translating this into the context of rotating maps means that, integrated over one full rotation from 0 to 2π, we want to maximize the number of visible labels. The consistency requirements of Been et al. [9] can immediately be applied for rotating maps.

OUR RESULTS. Initially, we define a model for rotating maps and show some basic properties of the different types of conflicts that may arise during rotation. Next, we prove that consistently labeling rotating maps is NP-hard, for the maximization of the total number of visible labels in one full rotation and NP-hard for the maximization of the visibility range of the least visible label. Finally, we present a new $1/4$-approximation algorithm and an efficient polynomial-time approximation scheme (EPTAS) for unit-height rectangles. A PTAS is called *efficient* if its running time is $O(f(\varepsilon) \cdot \text{poly}(n))$. Both algorithms can be extended to the case of rectangular labels with the property that the ratio of the smallest and largest width, the ratio of the smallest and largest height, as well as the aspect ratio of every label is bounded by a constant, even if we allow the anchor point of each label to be an arbitrary point of the label. This applies to most practical scenarios where labels typically consist of few and relatively short lines of text. For the case that a guaranteed optimal solution is preferred over the approximation produced by the EPTAS we propose an integer linear program (ILP) and a mixed integer linear program (MIP), both of which find the optimal solution. On the other hand, if there is only limited computing power available and the solution needs to be computed quickly, simple heuristics may provide solutions with sufficient quality. We propose three greedy heuristics, where one of them is a $1/8$-approximation for the case that only unit squares are used as labels. We conclude this chapter by giving an experimental evaluation of the proposed algorithms and models. First, we discuss the impact of the limitations which we have introduced by our model and then evaluate the performance of our algorithms.

related work. Most previous algorithmic research efforts on automated label placement cover *static* labeling models for point, line, or area features. For static point labeling, fixed-position models and slider models have been introduced [86, 136], in which the label, represented by its bounding box, needs to touch the labeled point along its boundary. The label number maximization problem is NP-hard even for the simplest labeling models, whereas there are efficient algorithms for the decision problem that asks whether all points can be labeled in some of the simpler models (see the discussion by Klau and Mutzel [131] or the comprehensive map labeling bibliography [186]). Approximation results [136, 2], heuristics [182], and exact approaches [131] are known for many variants of the static label number maximization problem. Additionally, to the label number maximization problem, there is also the label size maximization problem, where *all* labels need to be placed and the optimization goal is to find the largest factor by which all labels can be stretched such that not two labels overlap each other. The corresponding decision problem is NP-hard in the fixed-position model with four different positions [86], but is in P if only two positions per label are allowed[86]. As for the label size maximization problem, there are approximation algorithms [182, 86].

In recent years, *dynamic* map labeling has emerged as a new research topic that gives rise to many unsolved algorithmic problems. Petzold et al. [156] used a preprocessing step to generate a reactive conflict graph that represents possible label overlaps for maps of all scales. For any fixed scale and map region, their method computes a conflict-free labeling using heuristics. Mote [147] presents another fast heuristic method for dynamic conflict resolution in label placement that does not require preprocessing. The consistency desiderata of Been et al. [9] for dynamic labeling (no popping and jumping effects when panning and zooming), however, are not satisfied by either of the methods. Been et al. [10] showed NP-hardness of the label number maximization problem in the consistent labeling model and presented several approximation algorithms for the problem. Nöllenburg et al. [151] studied a dynamic version of the alternative boundary labeling model, in which labels are placed at the sides of the map and connected to their points by leaders. They presented an algorithm to precompute a data structure that represents an optimal one-sided labeling for all possible scales and thus allows continuous zooming and panning. None of the existing dynamic map labeling approaches supports map rotation. Recently, Gemsa et al. [100] discussed the problem of labeling a rotating map when only a small part of the map can be seen. This is a problem the manufacturers of modern GPS navigation devices face when implementing the algorithms for displaying the map. They present an NP-hardness proof as well as an approximation algorithm for unit squares. Finally, a recent paper investigated the label size maximization problem for rotating maps [189]. They show that the problem is in P if they consider unit height rectangles in the equivalent of the fixed position model with only one allowed position.

7.2 preliminaries

In this section we describe a general labeling model for rotating maps with axis-aligned rectangular labels and give a formal definition of the problem we investigate in this chapter.

Let M be an (abstract) map, consisting of a set $P = \{p_1, \dots, p_n\}$ of points in the plane together with a set $L = \{\ell_1, \dots, \ell_n\}$ of pairwise disjoint, closed, and axis-aligned rectangular labels in the plane. Each point p_i must coincide with a corner of its corresponding label ℓ_i; we denote that corner (and the point p_i) as the *anchor* of label ℓ_i. Since each label has four possible positions with respect to p_i this widely used model is known in the literature as the 4-position model (4P) [86]. *anchor*

As M rotates, each label ℓ_i in L must remain horizontally aligned and anchored at p_i. Thus, new label intersections form and existing ones disappear during the rotation of M.

We take the following alternative perspective on the rotation of M. Rather than rotating the points, say clockwise, and keeping the labels horizontally aligned we may instead rotate each label counterclockwise around its anchor point and keep the set of points fixed. It is easy to see that both rotations are equivalent in the sense that they yield exactly the same intersections of labels and occlusions of points.

We consider all rotation angles modulo 2π. For convenience we introduce the interval notation $[a,b]$ for any two angles $a,b \in [0,2\pi]$. If $a \le b$, this corresponds to the standard meaning of an interval, otherwise, if $a > b$, we define $[a,b] := [a,2\pi] \cup [0,b]$. For simplicity, we refer to any set of the form $[a,b]$ as an interval. We further define the length of an interval I as $|I| = b-a$ if $I = [a,b]$ with $a \le b$ and $|I| = 2\pi - a + b$ if $I = [a,b]$ with $a > b$.

A *rotation* of L is defined by a rotation angle $\alpha \in [0,2\pi)$. We define the rotated label set $L(\alpha)$ of all labels, each rotated by an angle of α around its anchor point. A *rotation labeling* of M is a function $\phi\colon L \times [0,2\pi) \to \{0,1\}$ such that $\phi(\ell,\alpha) = 1$ if label ℓ is visible or *active* in the rotation of L by α, and $\phi(\ell,\alpha) = 0$ otherwise. We call a labeling ϕ *valid* if, for any rotation α, the set of labels $L_\phi(\alpha) = \{\ell \in L(\alpha) \mid \phi(\ell,\alpha) = 1\}$ consists of pairwise disjoint labels. If two labels ℓ and ℓ' in $L(\alpha)$ intersect, we say that they have a (soft) *conflict* at α, i.e., in a valid labeling at most one of them can be active at α. We define the set $C(\ell,\ell') = \{\alpha \in [0,2\pi) \mid \ell \text{ and } \ell' \text{ are in conflict at } \alpha\}$ as the *conflict set* of ℓ and ℓ'. Further, we call the begin and end of a maximal contiguous conflict range in $C(\ell,\ell')$ *conflict events*.

rotation

rotation labeling

active

conflict

conflict events

For a label ℓ we call each maximal interval $I \subseteq [0,2\pi)$ with $\phi(\ell,\alpha) = 1$ for all $\alpha \in I$ an *active range* of label ℓ and define the set $A_\phi(\ell)$ as the set of all active ranges of ℓ in ϕ. We call an active range where both boundaries are conflict events a *regular* active range. Our optimization goal is to find a valid labeling ϕ that shows a maximum number of labels integrated over one full rotation from 0 to 2π. The value of this integral is called the *total activity* $t(\phi)$ and can be computed as $t(\phi) = \sum_{\ell \in L} \sum_{I \in A_\phi(\ell)} |I|$.

active range

regular

total activity

A valid labeling is not yet consistent in terms of the definition of Been et al. [9, 10]: for given fixed anchor points, labels clearly do not jump and the labeling is independent of the rotation history, but labels may still *flicker* multiple times during a full rotation from 0 to 2π, depending on how many active ranges they have in ϕ. In the most restrictive consistency model, which avoids flickering entirely, each label is either active for the full rotation $[0,2\pi)$ or never at all. We denote this model as *0/1-model*. However, this model is from our perspective too restrictive and we therefore propose a model where we say a rotation labeling is consistent if each label has only a single active range, which we denote here as the *1R-model*. We note that this can be immediately generalized to the *kR-model* that allows at most k active ranges for each label for $k \ge 2$. We denote the model that has no restrictions on the number of active ranges per label as ∞R model. However, only the 0/1-, and the 1R model follow the consistency criteria proposed by Been et al. [9, 10].

0/1-model

1R-model

kR-model

We apply another restriction to our consistency model, which is based on the occlusion of anchors. Among the conflicts in set $C(\ell,\ell')$ we further distinguish *hard conflicts*, i.e., conflicts where label ℓ intersects the anchor of label ℓ'. (In our definition every hard conflict is also a soft conflict.) If a labeling ϕ sets ℓ active during a hard conflict with ℓ', the anchor of ℓ' is occluded. This may be undesirable in some situation in practice, e.g., if every point in P carries useful information in the map, even if it is unlabeled. Thus we require that $\phi(\ell,\alpha) = 0$ during any hard conflict of a label ℓ with another label ℓ' at angle α.

hard conflicts

We distinguish the models further based on the way label-anchor conflicts are handled. We say a labeling model is a *hard-conflict* model, when no label may occlude an anchor point of another label, and otherwise (i. e., when a label may occlude any anchor) we denote it as *soft-conflict* model. This property of a labeling model is independent of the number of allowed active ranges per label, i. e., any combination of those two types of model is possible (e. g., 1R hard-conflict model, 3R soft-conflict model).

The objective in static map labeling is usually to find a maximum subset of pairwise disjoint labels, i. e., to label as many points as possible. Generalizing this objective to rotating maps means that integrated over all rotations $\alpha \in [0, 2\pi)$ we want to display as many labels as possible. This corresponds to finding the rotation labeling ϕ with maximum $t(\phi)$ over all rotation labelings in; we call this optimization problem MaxTotal. An alternative objective is to maximize over all rotation labelings ϕ the minimum length $\min\{|I| \in A_\phi(\ell) \mid \ell \in L\}$ of all active ranges in the 1R model; this problem is called MaxMin.

Although any combination of kR and hard/soft-conflict model is conceivable, we argue that there are only few sensible ones. We are mainly interested in the 1R hard-conflict model since it provides a middleground between the 0/1 model and the ∞R model, while keeping all anchor points on display during the full rotation. For the 1R hard-conflict model, we give complexity results, and approximation algorithms, as well as algorithmic approaches to obtain optimal solutions. In the experimental evaluation we compare the 1R model with other kR models, to assess the "cost" of restricting the maximum number of active ranges to 1 per label, and also evaluate the difference between the soft-conflict and hard-conflict model.

We note that our problem follows the constraints C1-C3 and rules R2 and is easily extended to follow the rule R3 proposed by Reimer and Rylov [161].

7.3 PROPERTIES OF ROTATION LABELINGS

In this section we show basic properties of rotation labelings. If two labels ℓ and ℓ' intersect in a rotation of α they have a (regular) *conflict* at α, i.e., in a consistent labeling at most one of them can be active at α.

We show the following lemma in a more general model, in which the anchor point p of a label ℓ can be *any* point within ℓ and not necessarily a point on the boundary $\partial\ell$ of the label ℓ:

Lemma 7.1 *For any two labels ℓ and ℓ' with anchor points $p \in \ell$ and $p' \in \ell'$ the set $C(\ell, \ell')$ consists of at most four disjoint contiguous conflict ranges.*

Proof. The first observation is that due to the simultaneous rotation of all initially axis-parallel labels in L, ℓ and ℓ' remain "parallel" at any rotation angle α. Rotation is a continuous movement and hence any maximal contiguous conflict range in $C(\ell, \ell')$ must be a closed "interval" $[\alpha, \beta]$, where $0 \leq \alpha, \beta < 2\pi$. Here we explicitly allow $\alpha > \beta$ by defining, in that case, $[\alpha, \beta] = [\alpha, 2\pi) \cup [0, \beta]$. At a rotation of α (resp. β) the two labels ℓ and ℓ' intersect only on their boundary. Let l, r, t, b be the left, right, top, and bottom sides of ℓ and let l', r', t', b' be the left, right, top, and bottom sides of ℓ' (defined at a rotation of 0). Since ℓ and ℓ' are parallel, the only possible cases, in which they intersect on their boundary but not in their interior are $t \cap b'$, $b \cap t'$, $l \cap r'$, and $r \cap l'$. Each of those four cases may appear twice, once for each pair of opposite corners contained in the intersection. Figure 7.2 shows all eight boundary intersection events. Each of the conflicts defines a unique rotation angle and obviously at most four disjoint conflict ranges can be defined with these eight rotation angles as their endpoints. □

In the following we look more closely at the conditions under which the boundary intersection events (the *conflict events*) occur and at the rotation angles defining them. Let h_t and h_b be the distances from p to t and b, respectively. Similarly, let w_l and w_b be the distances from p to l and r, respectively; see Figure 7.3. By h'_t, h'_b, w'_l, and w'_r we denote the corresponding values for label ℓ'. Finally, let d be the distance of the two anchor points p and p'. To improve readability of the following lemmas we define two functions $f_d(x) = \arcsin(x/d)$ and $g_d(x) = \arccos(x/d)$.

Lemma 7.2 *Let ℓ and ℓ' be two labels anchored at points p and p'. Then the conflict events in $C(\ell, \ell')$ are a subset of $\mathcal{C} = \{2\pi - f_d(h_t + h'_b), \pi + f_d(h_t + h'_b), f_d(h_b + h'_t), \pi - f_d(h_b + h'_t), 2\pi - g_d(w_r + w'_l), g_d(w_r + w'_l), \pi - g_d(w_l + w'_r), \pi + g_d(w_l + w'_r)\}$.*

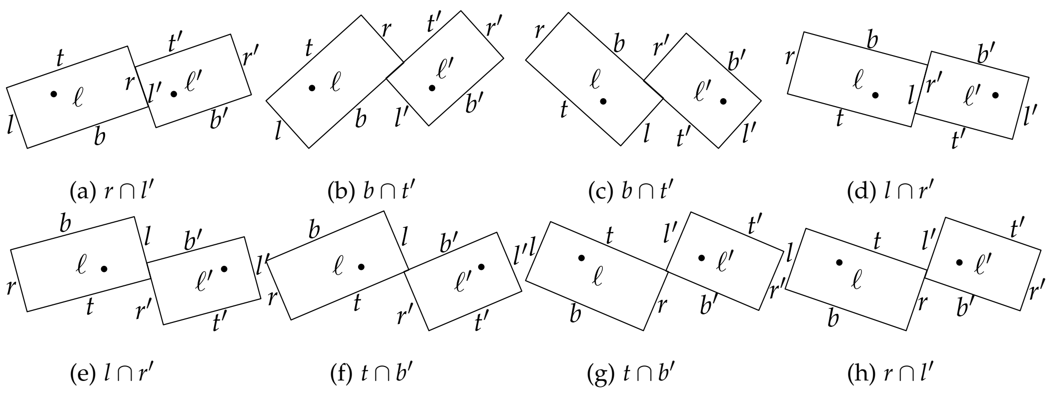


Figure 7.2: Two labels ℓ and ℓ' and their eight possible boundary intersection events. Anchor points are marked as black dots.

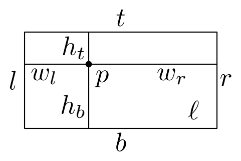


Figure 7.3: Parameters of label ℓ anchored at p.

Proof. Assume without loss of generality that p and p' lie on a horizontal line. First we show that the possible conflict events are precisely the rotation angles in $\mathcal{C}$. We start considering the intersection of the two sides t and b'. If there is a rotation angle under which t and b' intersect then we have the situation depicted in Figure 7.4 and by simple trigonometric reasoning the two rotation angles at which the conflict events occur are $2\pi - \arcsin((h_t + h'_b)/d)$ and $\pi + \arcsin((h_t + h'_b)/d)$. Obviously, we need $d \geq h_t + h'_b$. Furthermore, for the intersection in Figure 7.4(a) to be non-empty, we need $d^2 \leq (w_r + w'_l)^2 + (h_t + h'_b)^2$; similarly, for the intersection in Figure 7.4(b), we need $d^2 \leq (w_l + w'_r)^2 + (h_t + h'_b)^2$.

From an analogous argument we obtain that the rotation angles under which b and t' intersect are $\arcsin((h_b + h'_t)/d)$ and $\pi - \arcsin((h_b + h'_t)/d)$. Clearly, we need $d \geq h_b + h'_t$. Furthermore, we need $d^2 \leq (w_r + w'_l)^2 + (h_b + h'_t)^2$ for the first intersection and $d^2 \leq (w_l + w'_r)^2 + (h_b + h'_t)^2$ for the second intersection to be non-empty under the above rotations.

The next case is the intersection of the two sides r and l', depicted in Figure 7.5. Here the two rotation angles at which the conflict events occur are $2\pi - \arccos((w_r + w'_l)/d)$ and $\arccos((w_r + w'_l)/d)$. For the first conflict event we need $d^2 \leq (w_r + w'_l)^2 + (h_t + h'_b)^2$, and for the second we need $d^2 \leq (w_r + w'_l)^2 + (h_b + h'_t)^2$. For each of the intersections to be non-empty we additionally require that $d \geq w_r + w'_l$.

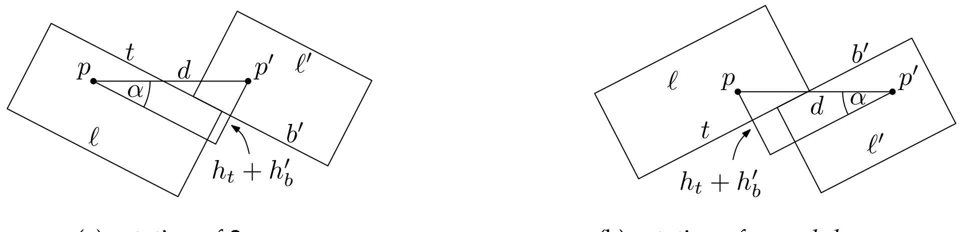
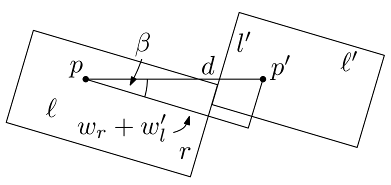
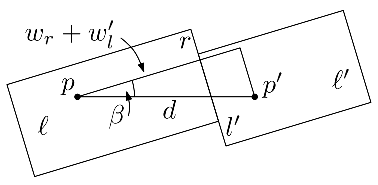


Figure 7.4: Boundary intersection events for $t \cap b'$.



(a) rotation of $2\pi - \beta$



(b) rotation of β

Figure 7.5: Boundary intersection events for $r \cap l'$.

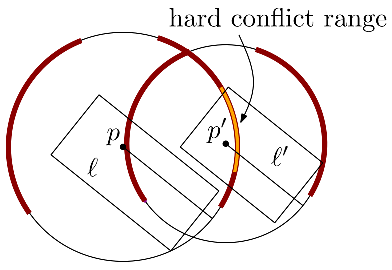


Figure 7.6: Conflict ranges of two labels ℓ and ℓ' marked in bold on the enclosing circles.

Similar reasoning for the final conflict events of $l \cap r'$ yields the rotation angles $\pi - \arccos((w_l + w'_r)/d)$ and $\pi + \arccos((w_l + w'_r)/d)$. The additional constraints are $d \geq w_l + w'_r$ for both events and $d^2 \leq (w_l + w'_r)^2 + (h_b + h'_t)^2$ for the first intersection and $d^2 \leq (w_l + w'_r)^2 + (h_t + h'_b)^2$. Thus, $\mathcal{C}$ contains all possible conflict events. □

One of the requirements for a valid labeling is that no label may contain a point in P other than its anchor point. For each label ℓ this gives rise to a special class of conflict ranges, called *hard* conflict ranges, in which ℓ may never be active. The rotation angles at which hard conflicts start or end are called *hard* conflict events. Every angle that is a (hard) conflict event is called a *label event*. Obviously, every hard conflict is also a regular conflict. Regular conflicts that are not hard conflicts are also called *soft* conflicts. We note that by definition regular conflicts are symmetric, i.e., $C(\ell, \ell') = C(\ell', \ell)$, whereas hard conflicts are not symmetric. The next lemma characterizes the hard conflict ranges.

Lemma 7.3 *For a label ℓ anchored at point p and a point $q \neq p$ in P, the hard conflict events of ℓ and q are a subset of $\mathcal{H} = \{2\pi - f_d(h_t), \pi + f_d(h_t), f_d(h_b), \pi - f_d(h_b), 2\pi - g_d(w_r), g_d(w_r), \pi - g_d(w_l), \pi + g_d(w_l)\}$.*

Proof. We define a label of width and height 0 for q, i.e., we set $h'_t = h'_b = w'_l = w'_r = 0$. Then the result follows immediately from Lemma 7.2. □

A simple way to visualize conflict ranges and hard conflict ranges is to mark, for each label ℓ anchored at p and each of its (hard) conflict ranges, the circular arcs on the circle centered at p and enclosing ℓ. Figure 7.6 shows an example.

In the following we show that the MaxTotal problem can be discretized in the sense that there exists an optimal solution whose active ranges are defined as intervals whose borders are label events. An active range *border* of a label ℓ is an angle α that is characterized by the property that the labeling ϕ is not constant in any ε-neighborhood of α. We call an active range where both borders are label events a *regular* active range.

Lemma 7.4 (Discretization Lemma) *Given a labeled map M there is an optimal rotation labeling of M consisting of only regular active ranges.*

Proof. Let ϕ be an optimal labeling with a minimum number of active range borders that are no label events. Assume that there is at least one active range border β that is no label event. Let α and γ be the two adjacent active range borders of β, i.e., $\alpha < \beta < \gamma$, where α and γ are active range borders, but not necessarily label events. Then let L_l be the set of labels whose active ranges have left border β and let L_r be the set of labels whose active ranges have right border β. For ϕ to be optimal L_l and L_r must have the same cardinality since otherwise we could increase the active ranges of the larger set and decrease the active ranges of the smaller set by an $\varepsilon > 0$ and obtain a better labeling.

So define a new labeling ϕ' that is equal to ϕ except for the labels in L_l and L_r: define the left border of the active ranges of all labels in L_l and the right border of the active ranges of all labels in L_r as γ instead of β. Since $|L_l| = |L_r|$ we shrink and grow an equal number of active ranges by the same amount. Thus the two labelings ϕ and ϕ' have the same objective value $\sum_{\ell \in L} |A_\phi(\ell)| = \sum_{\ell \in L} |A_{\phi'}(\ell)|$. Because ϕ' uses as active range borders one non-label event less than ϕ this number was not minimum in ϕ—a contradiction. As a consequence ϕ has only label events as active range borders. □

7.4 COMPLEXITY CONSIDERATIONS

In this section we show that finding an optimal solution for MaxTotal (and also MaxMin) is NP-hard in the 1R hard-conflict model even if all labels are unit squares and their anchor points are their lower-left corners. We present a gadget proof reducing from the NP-complete problem planar Planar 3Sat [139]. Before constructing the gadgets, we show a special property of unit-square labels.

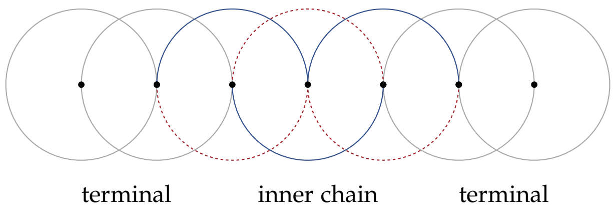
Lemma 7.5 *If two unit-square labels ℓ and ℓ' whose anchor points are their lower-left corners have a conflict at a rotation angle α, then they have conflicts at all angles $\alpha + i \cdot \pi/2$ for $i \in \mathbb{Z}$.*

Proof. Similar to the notation used in Section 7.3, let $f_d = \arcsin(1/d)$ and further let $g_d = \arccos(1/d)$. From Lemma 7.2 we obtain the set $\mathcal{C} = \{2\pi - f_d, \pi + f_d, f_d, \pi - f_d, 2\pi - g_d, g_d, \pi - g_d, \pi + g_d\}$ of conflict events for which it is necessary that the distance d between the two anchor points is $1 \leq d \leq \sqrt{2}$. Since $\arccos x = \pi/2 - \arcsin x$ the set $\mathcal{C}$ can be rewritten as $\mathcal{C} = \{f_d, \pi/2 - f_d, \pi/2 + f_d, \pi - f_d, \pi + f_d, 3\pi/2 - f_d, 3\pi/2 + f_d, 2\pi - f_d\}$. This shows that conflicts repeat after every rotation of $\pi/2$. □

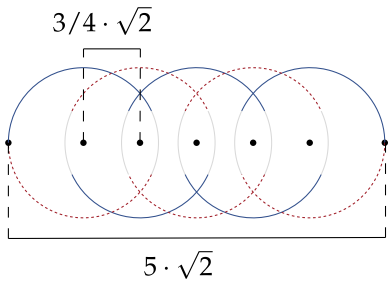
For every label ℓ we define the *outer circle* of ℓ as the circle of radius $\sqrt{2}$ centered at the anchor point of ℓ. Since the top-right corner of ℓ traces the outer circle we will use the locus of that corner to visualize active ranges or conflict ranges on the outer circle. Note that due to the fact that at the initial rotation of 0 the diagonal from the anchor point to the top-right corner of ℓ forms an angle of $\pi/4$ all marked ranges are actually offset by $\pi/4$.

The general idea behind the reduction is as follows. We describe how to transform a Planar 3Sat formula into an instance of MaxTotal that has a solution with value K if and only if the corresponding Planar 3Sat formula has a satisfying variable assignment. To do this we describe how to build *gadgets* that correspond to the variables and clauses of the Planar 3Sat formula. Since the gadgets need to properly correspond to Boolean variables the gadgets can have, in an optimal MaxTotal solution, only two possible states; one which corresponds to `true`, and the other to `false`. Similarly, the clause gadgets should contribute the same value to the total value of the optimal solution if and only if at least one of the variable gadgets is in the state `true`. Obviously, we need also some kind of connection between the variable gadgets and the clause gadgets to transmit the state of the variable gadgets. Those should contribute to the value of the optimal solution of MaxTotal the same value regardless of the state of the connected variable's state. If this is the case then we only need to choose the appropriate value K for which the value of the optimal solution of the instance indicates that the corresponding Planar 3Sat formula has a satisfying variable assignment. Hence, an exact polynomial time algorithm that solves MaxTotal can decide Planar 3Sat, which would imply NP = P.

We begin by describing basic building blocks from which we construct all our gadgets.



(a) Chain. The two outermost labels (grey circles) on each side are a terminal. A terminal can have several different states in an optimal solution.



(b) Inverter.

Figure 7.7: Depicted are the two basic building blocks chain and inverter. Each of them has only two states in any optimal solution of MaxTotal. One of those states is marked by the solid blue arcs, while the other is marked by the dotted red arcs

7.4.1 *Basic Building Blocks*

We base all our later gadgets on the following three basic structures.

chain CHAIN. A *chain* consists of at least four labels anchored at collinear points that are evenly spaced with distance $\sqrt{2}$. Hence, each point is placed on the outer circles of its neighbors. *terminals* *inner chain* We call the first and last two labels of a chain *terminals* and the remaining part *inner chain*, see Figure 7.7(a). We denote an assignment of active ranges to the labels as the *state* of the chain. The important observation is that in any optimal solution of MaxTotal an inner chain has only two different states, whereas terminals have multiple optimal states that are all equivalent for our purposes; see Figure 7.7(a). In particular, in an optimal solution each label of an inner chain has an active range of length π and active ranges alternate between adjacent labels. We will use the two states of chains as a way to encode truth values in our reduction.

Lemma 7.6 *In any optimal solution, any label of an inner chain has an active range of length π. The active ranges of consecutive labels alternate between $(0, \pi)$ and $(\pi, 2\pi)$.*

Proof. By construction every label has two hard conflicts at angles 0 and π, so no active range can have length larger than π. From Lemma 7.5 we know that every label has conflicts at $\pi/2$ and $3\pi/2$. These conflicts are soft conflicts and can be resolved by either assigning all odd labels the active range $(0, \pi)$ and all even labels the active range $(\pi, 2\pi)$ or vice versa. Obviously both assignments are optimal and there is no optimal assignment in which two adjacent labels have active ranges on the same side of π. □

For inner chains whose distance between two adjacent points is less than $\sqrt{2}$ the length of the conflict region changes, but the above arguments remain valid for any distance between 1 and $\sqrt{2}$.

INVERTER. The second basic building block is an *inverter*. It consists of five collinear labels that are evenly spaced with distance $3/4 \cdot \sqrt{2}$ as depicted in Figure 7.7(b). This means that the five labels together take up the same space as four labels in a usual inner chain. Similar to Lemma 7.6 the active ranges in an optimal solution also alternate. By replacing four labels of an inner chain with an inverter we can alter the parity of an inner chain.

TURN. The third building block is a *turn* that consists of four labels; see Figure 7.8. The anchor points p_a and p_b are at distance $\sqrt{2}$ and the pairwise distances between p_b, p_c, and

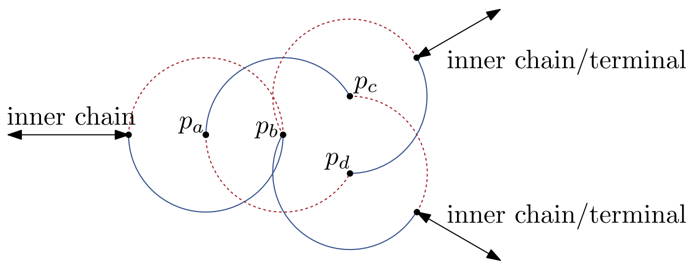


Figure 7.8: Turn. A turn that splits one inner chain into two inner chains. Like the chain, the turn has only two possible states in an optimal solution. One is marked by the solid blue arc, while the other is marked by the red dotted arc.

p_d are also $\sqrt{2}$ such that the whole structure is symmetric with respect to the line through p_a and p_b. The central point p_b is called *turn point*, and the two points p_c and p_d are called *outgoing points*. Due to the hard conflicts created by the four points we observe that the outer circle of p_b is divided into two ranges of length $5\pi/6$ and one range of length $\pi/3$. The outer circles of the outgoing points are divided into ranges of length π, $2\pi/3$, and $\pi/3$. The outer circle of p_a is divided into two ranges of length π. The outgoing points serve as connectors to terminals, inner chains, or further turns. Note, by coupling multiple turns we can divert an inner chain by any multiple of 30°.

Lemma 7.7 *A turn has only two optimal states and allows to split an inner chain into two equivalent parts in an optimal solution.*

Proof. We show that the optimal solution for the turn is $21/6\pi$ and that there are only two different active range assignments that yield this solution. Note that for the label ℓ_a the length of its active range is at most π. For ℓ_b it is at most $2/3\pi$ and for ℓ_c and ℓ_d it is at most π.

We first observe that ℓ_c and ℓ_d cannot both have an active range of length π since by Lemma 7.5 they have a soft conflict in the intersection of their length-π ranges. Thus at most one of them has an active range of length π and the other has an active range of length at most $5\pi/6$. But in that case the same argumentation shows that the active range of ℓ_b is at most $\pi/2$. Combined with an active range of length π for ℓ_a this yields in total a sum of $20\pi/6$.

On the other hand, if one of ℓ_c and ℓ_d is assigned an active range of length $2\pi/3$ and the other an active range of length π as indicated in Figure 7.8, the soft conflict of ℓ_b in one of its ranges of length $5\pi/6$ is resolved and ℓ_b can be assigned an active range of maximum length. This also holds for ℓ_a resulting in a total sum of $21\pi/6$.

Since the gadget is symmetric there are only two states that produce an optimal solution for the lengths of the active ranges. By attaching inner chains to the two outgoing points the truth state of the inner chain to the left is transferred into both chains on the right. □

7.4.2 *Gadgets of the Reduction*

The reduction is based on three different kinds of gadgets. One type for the variables, one for the clauses, and a gadget, which we call *pipe*, that connects the variable gadgets with the clauses. We now describe those gadgets and conclude this section by explaining how to position them. *pipe*

VARIABLE GADGET. The variable gadget consists of an alternating sequence of two building blocks: horizontal chains and *literal readers*. A literal reader is a structure that *literal readers*

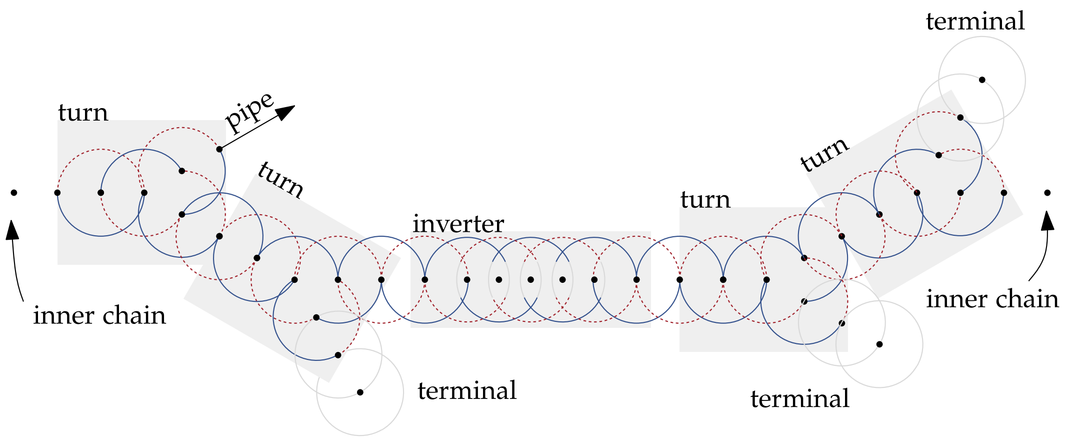


Figure 7.9: Literal Reader.

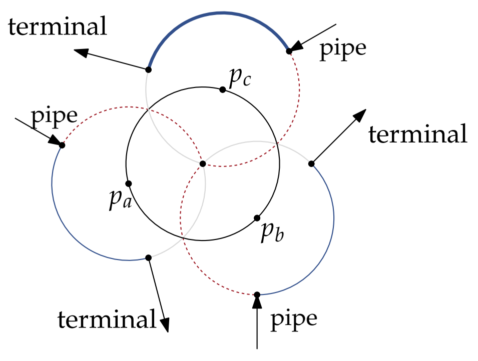
allows us to split the truth value of a variable into one part running towards a clause and the part that continues the variable gadget; see Figure 7.9. The literal reader consists of four turns, three terminals, and one modified inverter. The first turn connects to a literal pipe and the other three are dummy turns needed to lead the variable gadget back to our grid. Note that some of the distances between anchor points in the literal reader need to be slightly less than $\sqrt{2}$ in order to reach a grid point at the end of the structure. We can use an appropriately modified inverter to achieve this.

In order to encode truth values we define the state in which the first label of the first horizontal chain has active range $(0, \pi)$ as `true` and the state with active range $(\pi, 2\pi)$ as `false`.

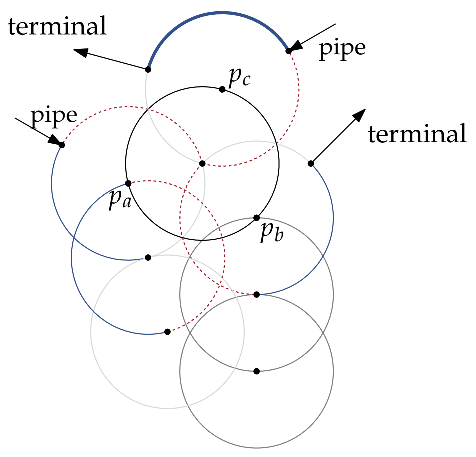
CLAUSE GADGET. The clause gadget consists of one *inner* and three *outer labels*, where the anchor points of the outer labels split the outer circle of the inner label into three equal parts of length $2\pi/3$; see Figures 7.10 and 7.11. Each outer label further connects to an incoming literal pipe and a terminal. These two connector labels are placed so that the outer circle of the outer label is split into two ranges of length $3\pi/4$ and one range of length $\pi/2$. Note that the terminals can be placed such that they cannot have a conflict with the other pipes or terminals; see Figure 7.10(b).

The general idea behind the clause gadget is as follows. The inner label obviously cannot have an active range larger than $2\pi/3$. Each outer label is placed in such a way that if it carries the value `false` it has a soft conflict with the inner label in one of the three possible active ranges of length $2\pi/3$; see Figure 7.11(a) for an illustration of this (for simplicity only the left outer label and the inner label are depicted). In this example the pipe coming from the left side transmits the truth value `false`. This forces the outer label ℓ_a with anchor p_a to have the active range indicated by the red dotted arc. This splits the arc from p_a to p_c of the inner label's outer circle into two arcs where one arc has length $\pi/2$ and the other $\pi/6$. The functionality of the other outer labels is identical. Hence, if all three pipes transmit the value `false` then every possible active range of the inner label of length $2\pi/3$ is affected by a soft conflict. Consequently, its active range can be at most $\pi/2$.

On the other hand, if at least one of the pipes transmits `true`, the inner label can be assigned an active range of maximum length $2\pi/3$. As can be seen in the Figures 7.11(b) and 7.11(c), when the outer label ℓ_a with anchor p_a has the active range indicated by the solid blue line (i. e., the attached pipe transmits the truth value `true`), no arc of the inner label is affected. The inner label can have active range that is corresponds to the arc from p_a to p_b of length $2\pi/3$.

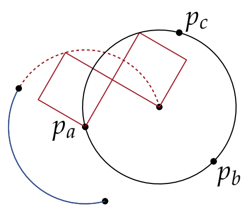


(a) Clause gadget with one inner and three outer labels.

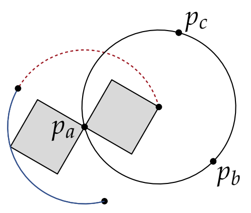


(b) Clause gadget with terminal the left inner label and attached pipe for the lower inner label. The labels of the terminal and the labels of the pipe are cannot intersect.

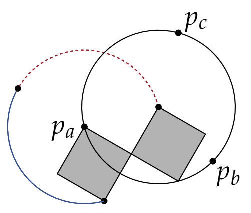
Figure 7.10: Illustration of the clause gadget.



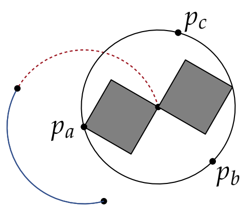
(a) Soft Conflict.



(b) Hard Conflict.



(c) Hard Conflict.



(d) Hard Conflict.

Figure 7.11: Illustration of the four conflicts of the left outer label with the inner label. The only soft conflict is depicted in (a). All remaining conflicts are hard conflicts.

Lemma 7.8 *There must be a label in a clause or in one of the incoming pipes with an active range of length at most $\pi/2$ if and only if all three literals of that clause evaluate to* `false`*.*

Proof. The active range for the lower-right outer label that is equal to the state `false` is $(3\pi/4, 3\pi/2)$. For the two other outer labels the active range corresponding to `false` is rotated by $\pm 2/3\pi$. Note that the outer clause labels can have an active range of at most $3/4\pi$ and the inner clause label can at most have an active range of at most $2/3\pi$. For every literal that is `false` one of the possible active ranges of the inner clause label is split by a conflict into two parts of length $\pi/2$ and $\pi/6$. This conflict is either resolved by assigning an active range of length $\pi/2$ to the inner clause label or by propagating the conflict into the pipe or variable where it is eventually resolved by assigning some active range with length at most $\pi/2$.

Otherwise, if at least one pipe transmits `true`, the inner label of the clause can be active for $2\pi/3$ while the outer clause labels have an active range of length $3\pi/4$ and no chain or turn has a label that is visible for less than $2\pi/3$. □

PIPES. Pipes propagate truth values of variable gadgets to clause gadgets. We use three different types of pipes, which we call *left arm*, *middle arm*, and *right arm*, depending on where the pipe attaches to the clause.

One end of each pipe attaches to a variable at the open outgoing label of a literal reader. Initially, the pipe leaves the variable gadget at an angle of 30°. By using sequences of turns, we can route the pipes at any angle that is an integer multiple of 30°. Thus we can make

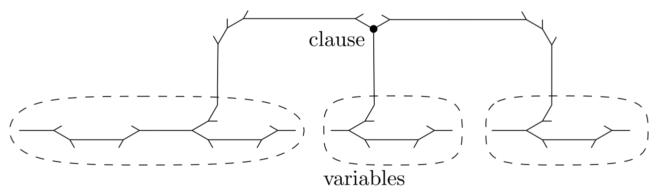


Figure 7.12: Sketch of the gadget placement for the reduction.

sure that for a clause above the variables the left arm enters the clause gadget at an angle of 150°, the middle arm at an angle of 270°, and the right arm at an angle of 30° with respect to the positive x-axis. For clauses below the variables the pipes are mirrored.

In order to transmit the correct truth value into the clause we first need to place the literal reader such that the turn point of the first turn corresponds to an even position in the variable chain. Next, for a positive literal we need a pipe of even length, whereas for a negative literal the pipe must have odd length. Note that we can always achieve the correct parity by making use of the inverter gadgets.

GADGET PLACEMENT. We place all variable gadgets on the same y-coordinate, and all clause gadgets and pipes lie below and above the variables and form three-legged "combs". The overall structure of the gadget arrangement is sketched in Figure 7.12.

At first glance it seems as if we might have a problem in describing the anchor coordinates with polynomially bounded precision, since, for example, each variable gadget requires us to place anchors with distance $\sqrt{2}$. By encoding the anchor coordinates with algebraic numbers, however, we can avoid this problem. It can be verified that all our constructed gadgets are either aligned with the coordinate axes directly or they are sloped by 15°, 30°, 45°, or 60° with respect to an axis-parallel line through a reference point. Now it is well known that the sine and cosine values of these angles can all be expressed as algebraic numbers in the number field $\mathbb{Q}(\sqrt{2}, \sqrt{3})$. Hence, if we choose, say, the leftmost anchor of the leftmost variable gadget as the origin, we can express the coordinates of any other anchor with respect to this origin as an algebraic number in $\mathbb{Q}(\sqrt{2}, \sqrt{3})$.

Theorem 7.1 MAXTOTAL *is NP-hard in the 1R hard-conflict model even if all labels are unit squares and their anchor points are their lower-left corners.*

Proof. For a given planar 3-SAT formula φ we construct the MAXTOTAL instance as described above. For this instance we can compute the maximum possible sum K of active ranges assuming that each clause is satisfiable. By Lemma 7.8 every unsatisfied clause forces one label to have an active range of only $\pi/2$. Thus we know that φ is satisfiable if and only if the MAXTOTAL instance has a total active range sum of at least K. Constructing and placing the gadgets can be done in polynomial time and space and the value of the optimal solution is a rational multiple of π.

There is one point left to argue. Recall that we require a MAXTOTAL instance to be an already labeled map, i. e., on the map no two labels intersect. We need to show that an instance of planar 3-Sat converted into a MAXTOTAL instance as described above fulfills this requirement. To see that this holds when all labels are placed in their initial position (i. e., at rotation angle 0) consider the following. Note that two labels of any gadget (except the inverter which we address later) intersect at exactly four rotation angles. Further, recall that the distance between two anchor points of any two labels in our construction that have an intersection is $\sqrt{2}$. This means that for two such labels to have an intersection when placed in their initial position it would require the anchor points to lie on a diagonal with slope 1. However, in our construction this does not exist. By simple calculations it is easy to see that that also the inverter gadget has no conflict when the labels are placed in their initial position. We can conclude that the constructed instance is a valid MaxTotal instance. Finally, we can conclude that MaxTotal is NP-hard. □

We note that the same construction as for the NP-hardness of MaxTotal can also be applied to prove NP-hardness of MaxMin. The maximally achievable minimum length of an active range for a satisfiable formula is $2\pi/3$, whereas for an unsatisfiable formula the maximally achievable minimum length is $\pi/2$ due to Lemma 7.8. This observation also yields that MaxMin cannot be efficiently approximated within a factor of $3/4$. We conclude the results in the following corollary.

Corollary 7.1 *MaxMin is NP-hard in the 1R hard-conflict model even if all labels are unit squares and their anchor points are their lower-left corners. Moreover, there exists no efficient approximation algorithm with an approximation factor larger than $3/4$ for this problem, unless P = NP.*

In the conference version of this chapter [104] (where we published preliminary results) we erroneously claimed that MaxTotal is contained in NP and thus the (corresponding decision) problem is NP-complete. Although it seems that due to the discretization lemma we can simply guess and verify a solution in polynomial time, we encounter the problem that in order to find the value of the guessed solution we potentially need to sum up irrational numbers. This, unfortunately, cannot be done in polynomial time on a Turing machine, and hence, we fail to show membership in
NP of MaxTotal.

7.5 APPROXIMATION ALGORITHMS FOR MAXTOTAL

In the previous section we have established that MaxTotal is NP-hard. Unless P = NP we cannot hope for an efficient exact algorithm to solve the problem. In the following we devise a $1/4$-approximation algorithm for MaxTotal and refine it to an EPTAS. For both algorithms we initially assume that labels are congruent unit-height rectangles with constant width $w \geq 1$ and that the anchor points are the lower-left corners of the labels. Let d be the length of the label's diagonal, i.e., $d = \sqrt{w^2 + 1}$.

Before we describe the algorithms we state four important properties that apply even to the more general labeling model, where anchor points are arbitrary points within the label or on its boundary, and where the ratio of the smallest and largest width and height, as well as the aspect ratio are bounded by constants:

(i) the number of anchor points contained in a rectangle is proportional to its area,
(ii) the number of conflicts a label can have with other labels is bounded by a constant,
(iii) any two conflicting labels produce only $O(1)$ conflict regions, and finally,
(iv) there is an optimal MaxTotal solution where the borders of all active ranges are events.

Properties (i) and (ii) are proved in Lemmas 7.9 and 7.10 using a simple packing argument. Property (iii) follows from property (ii) and Lemma 7.1. Property (iv) follows immediately from Lemma 7.4.

Lemma 7.9 *For any rectangle R with width W and height H, the number of anchor points in the interior or on the boundary of R is proportional to the area of R.*

Proof. Recall that by assumption all labels in the initial labeled map M are visible. Let the smallest label height be $h_{\min}$, the smallest label width be $w_{\min}$ and the smallest label area be $a_{\min}$. There can be at most $\lceil 2W/w_{\min} \rceil + \lceil 2H/h_{\min} \rceil$ independent labels intersecting the boundary of R such that their anchor points are contained in R. All remaining labels with an anchor point in R must be completely contained in R, i.e., there can be at most $\lceil W \cdot H/a_{\min} \rceil$ such labels. Hence, the number of anchor points in R is bounded by a constant. □

Lemma 7.10 *Each label ℓ has conflicts with at most a constant number of other labels.*

Proof. For two labels ℓ and ℓ' to have a conflict their outer circles need to intersect and thus the maximum possible distance between their anchor points is bounded by twice the maximum diameter of all labels in L. By the assumption that the height ratio, width ratio, and aspect ratio of all labels in L is bounded by a constant this diameter is constant. Hence we can define for each label ℓ a constant size area around its anchor point containing all relevant anchor points. By Lemma 7.9 this area contains only a constant number of anchor points. $\square$

7.5.1 *A* 1/4*-approximation for* MaxTotal

The basis for our algorithm is the *line stabbing* or *shifting* technique by Hochbaum and Maass [117], which has been applied before to *static* labeling problems for (non-rotating) unit-height labels [2, 136]. Consider a grid G where each grid cell is a square with side length $2d$. We can address every grid cell by its row and column index. Now we can partition G into four subsets by deleting every other row and every other column with either even or odd parity. Within each of these subsets we have the property that any two grid cells have a distance of at least $2d$. Thus no two labels whose anchor points lie in different cells of the same subset can have a conflict. For an illustration see Figure 7.13.

For a cell c we call the area that is not contained in c but within distance of at most d to c the *border* of c. To compute an optimal solution for the labels contained in c we must also consider anchor points that lie in this area since during a full rotation a label may intersect such an anchor point. We have depicted border of the cell in row 2 and column 3 in blue in Figure 7.13.

We say that a grid cell c *covers* a label ℓ if its anchor point lies inside c. By Lemma 7.9 only $O(1)$ labels are covered by a single grid cell. Combining this with Lemma 7.10 we see that the number of conflicts of the labels covered by the same cell is constant. It remains to determine the number of conflict events that are due to anchors in the border of the cell. It is easy to see that all anchors contained in the border of a cell lie inside a rectangle with dimensions $4d \times 4d$ centered at the cells center. By Lemma 7.9 we know that the number of anchor points is bounded by some constant and hence we can conclude that the number of conflict events contributed by these anchor points is also constant. This implies that the total number of conflict events per cell (cf. Lemma 7.4) is also constant.

The four different subsets of grid cells divide a MaxTotal instance into four subinstances, each of which decomposes into independent grid cells. If we solve all subsets optimally, at least one of the solutions is a 1/4-approximation for the initial instance due to the pigeon-hole principle.

Determining an optimal solution for the labels covered by a grid cell c works as follows. We compute, for the set of labels $L_c \subseteq L$ covered by c, the set E_c of label events (conflict events due to the soft/hard conflicts with labels covered by c and the conflict events due to hard conflicts with the anchors in the border of c). Due to Lemma 7.4 we know that there exists an optimal solution where all borders of active ranges are label events. Thus, to compute an optimal active range assignment for the labels in L_c we need to test all possible combinations of active ranges for all labels $\ell \in L_c$. For a single cell this requires only constant time.

We can precompute the non-empty grid cells by simple arithmetic operations on the coordinates of the anchor points and store those cells in a binary search tree. Since we have n anchor points there are at most n non-empty grid cells in the tree, and each of the cells holds a list of the covered anchor points. Building this data structure takes $O(n \log n)$ time and then optimally solving the active range assignment problem in the non-empty cells takes $O(n)$ time.

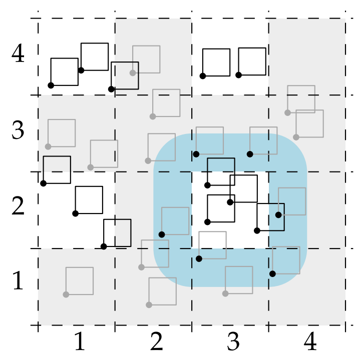


Figure 7.13: Example for the 1/4-approximation. Depicted are all grid cells and one of the four different subinstances (the even numbered rows and the odd numbered columns are selected). The blue area depicts the area in which the anchor points lie which may be important for the labels in the cell the blue area surrounds.

Theorem 7.2 *There exists an $O(n \log n)$-time algorithm that yields a $1/4$-approximation of* MAXTOTAL *for congruent unit-height rectangles with their lower-left corners as anchor points.*

7.5.2 *An Efficient Polynomial-Time Approximation Scheme for* MAXTOTAL

We extend the technique for the $1/4$-approximation to achieve a $(1-\varepsilon)$-approximation. Let again G be a grid whose grid cells are squares of side length $2d$. For any integer k we can remove every k-th row and every k-th column of the grid cells, starting at two offsets i and j ($0 \leq i, j \leq k-1$). This yields collections of meta cells of side length $(k-1) \cdot 2d$ that are pairwise separated by a distance of at least $2d$ and thus independent. In total, we obtain k^2 such collections of meta cells. For an illustration see Figure 7.14.

For a given $\varepsilon \in (0,1)$ we set $k = \lceil 2/\varepsilon \rceil$. Let c be a meta cell for the given k and let again L_c be the set of labels covered by c, and E_c the set of label events for L_c. Then, by Lemmas 7.9 and 7.10, the number of labels $|L_c|$ inside c is in $O(1/\varepsilon^2)$ and thus, the number of conflicts events $|E_c|$ for the all labels inside c is in $O(1/\varepsilon^4)$. The same number of conflict events is due to anchor points in the border of the meta cell c. Hence, for each label there are $O(1/\varepsilon^8)$ possible combinations of conflict events. Since we need to test all possible active ranges for all labels in L_c, it takes $O(2^{O(1/\varepsilon^2 \log 1/\varepsilon^8)})$ time to determine an optimal solution for the meta cell c.

For a given collection of disjoint meta cells we determine (as in Section 7.5.1) all $O(n)$ non-empty meta cells and store them in a binary search tree such that each cell holds a list of its covered anchor points. This requires again $O(n \log n)$ time. So for one collection of meta cells the time complexity for finding an optimal solution is $O(n 2^{O(1/\varepsilon^2 \log 1/\varepsilon^8)} + n \log n)$. There are k^2 such collections and, by the pigeon hole principle, the optimal solution for at least one of them is a $(1-\varepsilon)$-approximation of the original instance. This yields the following theorem.

Theorem 7.3 *There exists an EPTAS that computes a $(1-\varepsilon)$-approximation of* MAXTOTAL *for congruent unit-height rectangles with their lower-left corners as anchor points. Its time complexity is $O((n 2^{O(1/\varepsilon^2 \log 1/\varepsilon)} + n \log n)/\varepsilon^2)$.*

We note that this EPTAS basically relies on properties (i)–(iv) and that there is nothing special about congruent rectangles anchored at their lower-left corners. Hence we can generalize the algorithm to the more general labeling model, in which the ratio of the label heights, the ratio of the label widths, and the aspect ratios of all labels are bounded by constants. Furthermore, the anchor points are not required to be label corners; rather they can be any point on the boundary or in the interior of the labels. Finally, we can even ignore

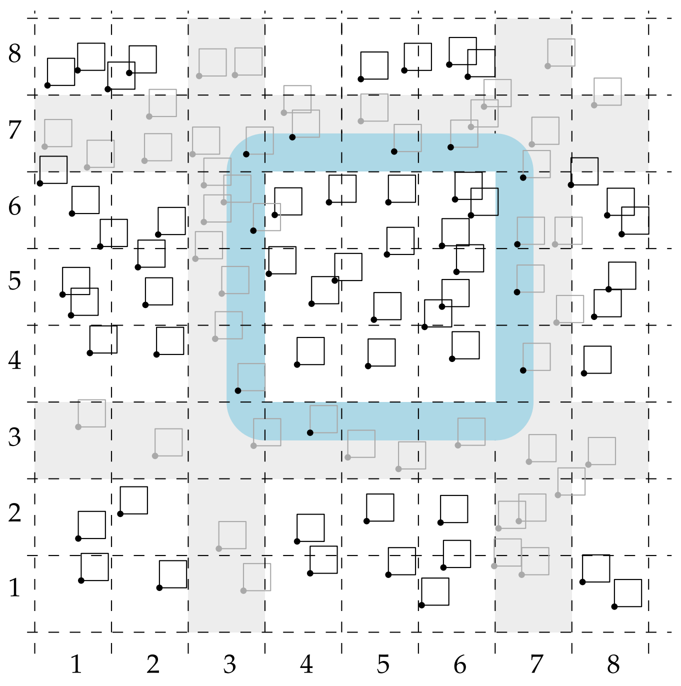


Figure 7.14: Illustration for our EPTAS for $k = 4$. The blue area is the border of the meta cell spanning columns and rows 4-6.

the distinction between hard and soft conflicts, i.e., allow that anchor points of non-active labels are occluded. Properties (i)–(iv) still hold in this general model. The only change in the EPTAS is to set the width and height of the grid cells to twice the maximum diameter of all labels in L.

Corollary 7.2 *There exists an EPTAS that computes a $(1 - \varepsilon)$-approximation of* MaxTotal *in the general labeling model with rectangular labels of bounded height ratio, width ratio, and aspect ratio, where the anchor point of each label is an arbitrary point in that label. The time complexity of the EPTAS is $O((n2^{O(1/\varepsilon^2 \log 1/\varepsilon)} + n \log n)/\varepsilon^2)$.*

7.6 heuristics for maxtotal

In this section we describe three greedy algorithms for solving MaxTotal. These algorithms are conceptually simple, and easy to implement. However, in general we cannot give quality guarantees for the solutions obtained by the algorithms.

All proposed algorithms follow the same principle. Throughout its execution, the algorithm maintains a set L' of label candidates that contains all labels it can still choose to assign an active range to. Further, for each label in L' the algorithm maintains the label's *maximum active range*. The maximum active range of a label ℓ is the single longest contiguous interval in $[0, 2\pi)$ (which may span over 2π) that ℓ can have as active range such that i) it does not overlap an anchor of another label in the instance (hard conflict), and ii) the label's maximum active range does not include a rotation angle α that is included in the active range of another label (already considered by the algorithm) that has a conflict with ℓ at α (soft conflict).

The algorithm initializes the set L' with all labels of the instance and computes each label's maximum active range (at initialization it only needs to consider hard conflicts). In each step of the algorithm it first selects and removes a label ℓ from L', assigns it its maximum active range, and finally updates those labels in L' whose active range is affected by assigning ℓ its maximum active range. The three algorithms differ only in the criteria that determines which label is selected from L'.

The first algorithm we propose is called GreedyMax. In each step the algorithm selects the label with the longest maximum active range among all labels left in L'. Ties are broken arbitrarily. The second algorithm, GreedyLowCost, determines for the maximum active range for each label the cost of adding it to the solution. This means that while considering a label $\ell_i \in L$ with maximum active range $A(\ell)$ the algorithm determines for all labels in L' that are in conflict with ℓ_i during $A(\ell)$ by how much their maximum active range would shrink. The sum of this is the *cost* of choosing ℓ_i with its active range $A(\ell)$. Among all labels GreedyLowCost chooses the label which has the lowest cost associated with it. Finally, the last algorithm, GreedyBestRatio is a combination of the two preceding ones. In each step the algorithm chooses the label ℓ_i with maximum active range $A(\ell)$ whose ratio of the length $|A(\ell)|$ to its cost is maximum among all remaining labels in L'.

Theorem 7.4 *In the* 1*R-model the algorithm GreedyMax has time complexity $O(cn(c + \log n))$ and the algorithms GreedyLowCost and GreedyBestRatio have time complexity $O(cn(c^2 + \log n))$, where n is the number of labels and c is the maximum number of conflicts per label in the input instance. The space consumption of all algorithms is in $O(cn)$.*

Proof. We begin by describing the time complexity for GreedyMax and sketch how to adapt the proof for GreedyBestRatio and GreedyLowCost.

In each step of the algorithm it first selects the label ℓ with maximum active range still left in L'. Then, it needs to update the maximum active range of those labels in L' that have a conflict with ℓ. To do this efficiently we maintain a maximum heap H that contains all labels in L' with the length of their maximum active range as key. This allows us to find and remove the label with maximum active range in $O(\log n)$ time. To determine the new maximum active ranges for those labels in L' that are in conflict with ℓ, we conduct a simple linear sweep over $[0, 2\pi)$. Note that there are at most $O(c)$ conflict events and hence a single sweep requires $O(c)$ time. Finally, we need to update the value of the maximum active ranges in the global heap. This requires $O(c \log n)$ time. Thus, a single step in the algorithm requires $O(c^2 + c \log n)$ time. Since there are n steps, the claimed running time follows.

For GreedyLowCost and GreedyBestRatio we use the same approach, but we store the labels with their cost and gain-cost ratio as key, respectively. The main difference compared to GreedyMax is the necessity to compute the cost of selecting a maximum active range, which can be done straightforwardly in $O(c^2)$ time per label. After selecting a label and assigning it its maximum active range as active range, the maximum active range of at most $O(c)$ labels still left in L' may change. For these labels, the algorithm needs to recompute the cost of selecting their maximum active ranges and update these values in the maximum heap. In total a single step for GreedyLowCost or GreedyBestRatio requires $O(c^3 + c \log n)$ time. The claimed time complexity follows. The required space is dominated by the storage required to store for each label its relevant conflict events. This takes $O(cn)$ space. □

By using a more efficient encoding of the maximal disjoint intervals that can be assigned to label ℓ_i in a heap, the running time of GreedyMax can be further improved to $O(cn \log n)$.

Theorem 7.5 *GreedyMax can be implemented with time complexity $O(cn \log n)$ and space requirement in $O(cn)$.*

Proof. Similar to the proof of Theorem 7.4, we maintain a global heap that contains the maximum possible active range for each label. In the following we describe the modified representation of the possible active ranges for the individual labels.

We maintain for *each* label $\ell_i \in L$ a maximum heap H_i that maintains all maximal disjoint intervals the label ℓ_i can be assigned as active range. Further, *each* label also maintains a balanced binary tree T_i on the same set of intervals, i. e., we store the left and right endpoints of the intervals in T_i. Should one of the intervals span over 2π, we split it into two intervals at 2π. Since there are $O(c)$ conflict events per label, both T_i and H_i can contain at most $O(c)$ elements.

Now, when the algorithm has chosen a label ℓ with maximum active range $A(\ell)$ with left and right endpoints a and b, respectively, it needs to update the maximum active range of each label ℓ_i that is in conflict with ℓ. So, for each label ℓ_i in conflict with ℓ, we query ℓ_i's binary search tree T_i with a and b. This allows us to obtain all k maximal intervals of ℓ_i that partly overlap with, or are completely contained in $A(\ell)$. The intervals completely contained in $A(\ell)$ need to be removed from T_i and H_i, while the intervals (at most two) that are only partly contained in $A(\ell)$ are shrunk or split.

The query on the binary search tree requires $O(k + \log c)$ time, where k is the number of reported intervals. Updating both T_i and H_i requires then $O(\log c)$ time for the intervals that get shrunk (or split). Deleting the elements is more costly, but since in both the binary tree as well as the heap there can be at most $O(c)$ elements, and we insert and remove each element at most once, we can conclude that inserting and deleting requires per binary tree and heap each in total at most $O(c \log c)$ time. Since there are $O(n)$ heaps and trees, this requires in total $O(cn \log c)$ time.

We now summarize these results. In each step we require $O(\log n)$ time to determine and remove the label ℓ with maximum active range from the global heap. Then, we shrink or split the intervals of labels that are in conflict with ℓ in $O(c \log c)$ time. We need to update the global heap since the maximum active range of the $O(c)$ labels that are in conflict with ℓ might have changed. This update requires $O(c \log n)$ time. Finally, we need to repeat this step $O(n)$ times, yielding $O(cn(\log c + \log n)) = O(cn \log n)$ time. As stated above, the insertion and deletion into the label's own binary trees and maximum heaps requires in total $O(cn \log n)$ time, yielding a total time complexity of $O(cn \log n)$. The space consumption is dominated by the $O(n)$ trees, each of size $O(c)$. We conclude that the algorithm requires $O(cn)$ space. □

approximation for unit square labels. While we claimed at the beginning of the section that in general we cannot prove any quality criteria for the presented algorithms for the special case that labels are unit squares we can show that GreedyMax is an approximation algorithm with approximation ratio $1/8$. This is due to a result by Gemsa et al. [100, Lemma 1 and Lemma 2] in which the authors consider a similar problem. The results of both Lemmas imply the following.

Lemma 7.11 ([100]) *For any unit square label ℓ there is no rotation angle $\alpha \in [0, 2\pi)$ for which there are more than eight unit square labels that overlap with ℓ and that are pairwise do not overlap.*

Since in each step of the GreedyMax we select the label ℓ with maximum active range we can conclude by Lemma 7.11 that the cost of choosing ℓ is at most eight times of its maximum active range. We summarize this in the following.

Corollary 7.3 *For the special case that we restrict the input of* MaxTotal *to consist only of unit square labels, the algorithm GreedyMax is a $1/8$-approximation algorithm.*

7.7 finding optimal solutions

In this section we describe two approaches to solve MaxTotal and one approach to solve MaxMin optimally by using (mixed) integer linear program formulations. The first integer linear program (ILP) described in Section 7.7.1 is conceptually simple and relies on the discretization lemma (Lemma 7.4). Unfortunately, the formulation does not allow an easy adaption such that it can solve MaxMin. The second approach (see Section 7.7.2) is a mixed integer linear program (MIP), i. e., a linear program with both integer and continuous variables, that we can use (with minor adjustments) to solve both MaxTotal and MaxMin. However, it is also significantly more complicated.

7.7.1 *An Integer Linear Program for* MaxTotal

In Section 7.9 we have seen how to construct an efficient polynomial-time approximation scheme for MaxTotal. One key ingredient for the EPTAS was to consider only small square sections in the plane in which (due to the requirement on the input) only few labels can be contained. Although from a theoretical point of view, finding the optimal solution for such a small instance requires only constant time by simply trying all possible combinations of possible solutions, the running time of such a brute force algorithm is, in practice, prohibitively high. Since we ware still interested in finding solutions to the problem we propose a formulation of the problem as integer linear program (ILP). This formulation is very similar to that of Gemsa et al. [100] but simpler in some respects.

The key idea of the ILP presented here is to determine regular active ranges induced by the ordered set of all conflict events. Our model contains for each label ℓ and each interval I a binary decision variable, which indicates whether or not ℓ is active during I. We add constraints to ensure that (i) no two conflicting labels are active at the same time within their conflict range and (ii) at most k disjoint contiguous active ranges can be selected for each label as required in the kR-model.

MODEL. For simplicity we assume in this section that the length of each conflict range is strictly larger than 0. This assumption is not essential for our ILP formulation, but makes a description much easier.

Let E be the ordered set of conflict events that also contains 0 and 2π, and let $E[j]$ be the interval between the j-th and the $(j+1)$-th element in E. We call such an interval $E[j]$ an *atomic interval* and always consider its index j modulo $|E|-1$. For each label $\ell_i \in L$ and for each atomic interval $E[j]$ we introduce two binary variables x_i^j and b_i^j to our model. We refer to the variables of the form x_i^j as *activity variables*. The intended meaning of x_i^j is that its value is 1 if and only if the label ℓ_i is active during the j-th atomic interval; otherwise x_i^j has value 0. We use the binary variables b_i^j to indicate the start of a new active range and to restrict their total number to 1. This is achieved by adding the following constraints to our model.

$$x_i^j - b_i^j \leq x_i^{j-1} \qquad \forall \ell_i \in L \quad \forall j \in \{0, \ldots, |E|-2\} \tag{7.1}$$

$$\sum_{0 \leq j \leq |E|-2} b_i^j \leq 1 \qquad \forall \ell_i \in L \tag{7.2}$$

The effect of constraint (7.1) is that it is only possible to start a new active range for label ℓ_i with atomic interval $E[j]$ (i.e., $x_i^{j-1} = 0$ and $x_i^j = 1$) if we account for that range by setting $b_i^j = 1$.

Due to constraint (7.2) this can happen at most k times per label. We can also allow arbitrarily many active ranges per label as in the ∞R-model by completely omitting the variables b_i^j and the above constraints.

It remains to guarantee that no two labels can be active when they are in conflict. This can be done straightforwardly since we can compute for which atomic intervals two labels are in conflict and we ensure that not both activity variables can be set to 1. More specifically, for every pair of labels ℓ_i, ℓ_k and for every atomic interval j during which they are in conflict, we add the constraint

$$x_i^j + x_k^j \leq 1. \tag{7.3}$$

Incorporating hard conflicts can also be done easily as a hard conflict simply excludes certain atomic intervals from being part of an active range. We determine for each label all such atomic intervals in a preprocessing step and set the corresponding activity variables to 0.

Among all feasible solutions that satisfy the above constraints, we maximize the objective function $\sum_{\ell_i \in L} \sum_{0 \le j \le |E|-2} x_i^j \cdot |E[j]|$, which is equivalent to the total activity $t(\phi)$ of the induced labeling ϕ.

This ILP considers only regular active ranges, since label activities change states only at conflict events. However, by Lemma 7.4, there always exists an optimal solution that is regular, and hence we are guaranteed to find a globally optimal solution.

We have now given a complete description of our ILP model, and now turn towards the analysis of the number of variables and constraints necessary for our model. Let e and c be the number of conflict events and the maximum number of conflict events per label in a MaxTotal instance, respectively. In the worst case the number of constraints that ensure that the solution is conflict-free (i. e., constraint (7.3)) is in $O(c \cdot e)$ per label, whereas we require only $O(e)$ constraints of the other two types of constraints per label.

Theorem 7.6 *The ILP (1)–(3) solves* MaxTotal *and has at most $O(e \cdot n)$ variables and $O(c \cdot e \cdot n)$ constraints, where n is the number of labels, e the number of conflict events, and c the maximum number of conflicts per label.*

EXTENSIONS. In Section 7.2 we explained that, in order to reduce flickering, we require that each label has at most one active range. However, we might be able to reduce flickering even more by finding the among all optimal solutions one that has the fewest active ranges. We can modify our ILP to accommodate for this by modifying the objective function slightly. Let $s > 0$ denote the length of a shortest atomic interval (recall that all atomic intervals are assumed to have positive length). Thus, whenever a label is active during an atomic interval, the total activity increases by at least s. To minimize the number of active ranges, we substract $s/2$ from the objective function for each active range. In this way, a solution with larger total activity is always preferred over a solution with less total activity, while among two solutions with the same total activity the one with fewer active ranges has the greater objective value. Hence, to minimize the number of active ranges while maintaining optimality, we modify the objective function to $\sum_{\ell_i \in L} \sum_{0 \le j \le |E|-2} \left(x_i^j \cdot |E[j]| - b_i^j \cdot s/2 \right)$. To ensure that if label ℓ_i is active at any time, then at least one of the variables b_i^j is 1, we also add the following constraints.

$$(|E|-1) \cdot \sum_{j=0}^{|E|-2} b_i^j \ge \sum_{j=0}^{|E|-2} x_i^j \qquad \forall \ell_i \in L \tag{7.4}$$

Without this constraint, it would be possible to have a label ℓ_i active for the whole range $[0, 2\pi)$ but with $b_i^j = 0$ for all $j \in \{0, \dots, |E|-2\}$. We note that it follows from the proof of Lemma 7.4 that also for the modified problem there always exists an optimal solution that is regular. Hence an optimal solution to the above ILP is indeed a global optimum.

As stated before we argued that to reduce flickering we require that each label is active in at most one contiguous range (1R model). However, it might be worthwhile to explore the optimal solution for the case that each label may be active for k active ranges, where $k \ge 2$. This can be easily included in the ILP by changing constraint (7.2) to

$$\sum_{0 \le j \le |E|-1} b_i^j \le k \qquad \forall \ell_i \in L. \tag{7.5}$$

Finally, we can adapt this formulation such that it can handle *weighted* labels when we make the reasonable assumption that the weight of a label is the length of its activity multiplied with a label specific constant w_ℓ. To allow this we only need to make a minor modification to the objective function.

7.7.2 *A Mixed Integer Linear Program for* MaxTotal *and* MaxMin

In Section 7.7.1 we have already given a simple integer linear programming formulation with which we can obtain an optimal solution to a MaxTotal instance. This formulation relied heavily on the discretization lemma, which we cannot use for MaxMin and hence, we fail to solve MaxMin with this approach.

In the following we give a mixed integer linear programming (MIP) formulation, that allows us to find an optimal solution for MaxTotal, and with only a minor modifications, it allows us to find an optimal solution for MaxMin. We begin by describing the general idea behind our approach, and proceed by describing the necessary variables and constraints for our model.

7.7.2.1 *Preliminaries*

The general idea behind this approach is somewhat more intuitive than the one presented before. In this model, we will determine the endpoints of each label's active range directly. For simplicity, we "unroll" the interval $[0, 2\pi)$ to $[0, 4\pi)$. This allows us to have active ranges that are "proper" intervals, in that the active range's left endpoint is always smaller than its right endpoint. To allow this we need to modify the definitions of Section 7.2 slightly. The domain of the rotation labeling function ϕ is extended to $L \times [0, 4\pi)$, and accordingly the set $A_\phi(\ell)$. For simplicity we refer in the following the active range of the label ℓ_i by $A(\ell_i)$, and we denote the left endpoint of its active range $A(\ell_i)$ by a_i and the right endpoint by b_i. For this section we assume that the active ranges are open intervals, i. e., $A(\ell_i) = (a_i, b_i)$. We discuss the implication of this at the end of this section. Finally, we assume in this section that when we talk about conflict ranges that these conflict ranges are maximal.

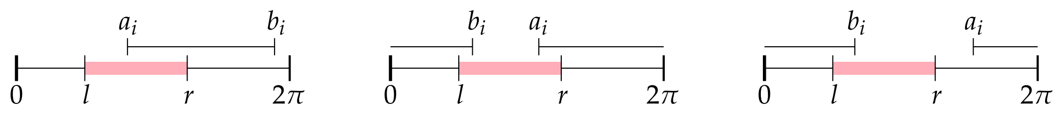
For this mixed integer linear programming formulation we will treat each conflict range between two labels separately. Conflict ranges that span over 2π are split into two separate conflict ranges at 2π. Since we have unrolled the interval $[0, 2\pi)$ to $[0, 4\pi)$ each conflict range appears twice. More specifically, a conflict between two labels with conflict range $[l, r]$ appears again at $[l + 2\pi, r + 2\pi]$. We refer to the conflict range contained in $[0, 2\pi)$ as *left conflict range* and its copy in $[2\pi, 4\pi)$ as *right conflict range*.

The critical part of this formulation is to avoid that two labels are active while they are in conflict, i. e., handling soft conflicts. The general idea behind the approach is to detect whenever the active range of any label ℓ_i has a non-empty intersection with the conflict set $C(\ell_i, \ell_j)$ for any ℓ_j. When this is detected certain constraints are "activated" which ensures that the intersection of ℓ_j's active range with $C(\ell_i, \ell_j) \cap A(\ell_i)$ is empty, i. e., not both ℓ_i and ℓ_j are active during a rotation for which both labels are in conflict with each other. In the following we discuss how to handle soft conflicts between two labels. This description is somewhat decoupled from the actual MIP formulation, but it is necessary to understand the constraints we require for the MIP formulation which we give later.

SOFT CONFLICTS. Consider a label ℓ_i with active range $A(\ell_i) = (a_i, b_i)$. Further, let ℓ_i be in conflict with another label ℓ_j during the conflict range $[l, r]$. If the intersection of $A(\ell_i)$ with $[l, r]$ (or the corresponding right conflict range $[l + 2\pi, r + 2\pi]$) is non-empty we can distinguish between three cases:

I) $a_i < r$ and $b_i > l$,
II) $a_i < r$ and $b_i > l + 2\pi$,
III) $a_i \geq r$ and $b_i > l + 2\pi$.

At first glance it seems strange that we distinguish between three cases, since case II) directly implies that case I) is true. We come back to this seemingly obvious flaw in our formulation in Section 7.7.2.2 after we have given a full description of our ILP formulation. However, to



(a) Case I. (b) Case II. (c) Case III.

Figure 7.15: Illustration of the three different cases for non-unrolled intervals. The interval shaded in light-red is a conflict interval between two labels ℓ_i and ℓ_j.

Table 7.1: Cases and the constraints under which they pose no problem.

Case	Constraints					
I	E1	E2	E3	E4	E5	E6
II	E1	E5				
III	E1	E5	E7			

give a first hint that these cases really are different in some respect we refer the reader to Figures 7.15 where we show examples for the three cases.

We observe that if and only if one of the above cases applies, the intersection of (a_i, b_i) with $[l, r]$ (or its corresponding right conflict interval) is non-empty. We now describe for each case which conditions must be met by the active range of the second label ℓ_j such that not both ℓ_i and ℓ_j are active when they are in conflict. For a examples for all cases see Figures 7.16(a)–7.16(d).

Should case I) apply, i. e., $a_i < r$ and $b_i > l$ then not both labels ℓ_i and ℓ_j are active while in conflict if (at least) one of the following conditions applies to ℓ_j's active range (a_j, b_j).

E1) ℓ_j's active range ends before the left conflict range begins (i. e., $b_j \leq l$)

E2) ℓ_j's active range ends before ℓ_i's active range begins (i. e., $b_j \leq a_i$)

E3) ℓ_j's active range begins after the end of ℓ_i's active range and it ends before the right conflict range (i. e., $a_j \geq b_i$ and $b_j \leq l + 2\pi$)

E4) ℓ_j's active range begins after the end of ℓ_i's active range and it ends before the begin of ℓ_i's conflict range mirrored into the second part of $[0, 4\pi)$ (i. e., $a_j \geq b_i$ and $b_j \leq a_i + 2\pi$)

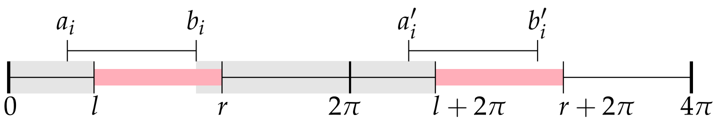
E5) ℓ_j's active range begins after the end of the conflict range and it ends before the right conflict range (i. e., $a_j \geq r$ and $b_j \leq l + 2\pi$)

E6) ℓ_j's active range begins after the end of the conflict range and it ends before the begin of ℓ_i's conflict range mirrored into the second part of $[0, 4\pi)$ (i. e., $a_j \geq r$ and $b_j \leq a_i + 2\pi$)

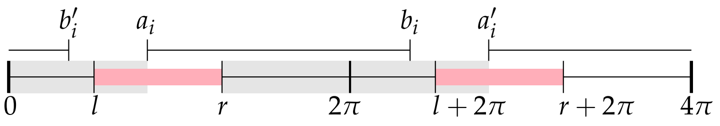
We further observe that should case II) apply, it is necessary and sufficient if either E1 or E5 is true. Finally, should case III) apply, both labels ℓ_i and ℓ_j cannot be active at a rotating angle under which they are in conflict if and only if E1 or E5 or the following condition E7 is true:

E7) ℓ_j's active range begins after the end of ℓ_i's active range mirrored into the first part of $[0, 4\pi)$ and ends before the begin of the right conflict range (i. e., $a_j \geq b_i - 2\pi$ and $b_j \leq l + 2\pi$).

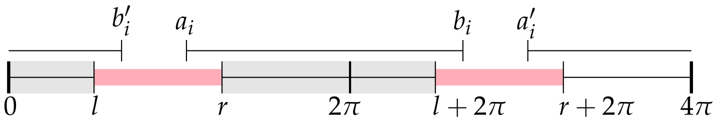
We have compiled an overview which cases may be resolved by which constraints in Table 7.1. We are now ready to discuss our model in more detail, i. e., the necessary variables and constraints which we need to add to solve MaxTotal/MaxMin.



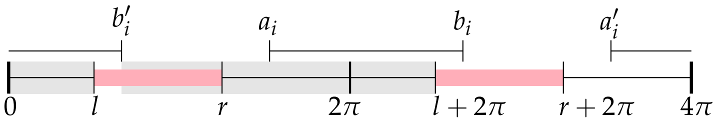
(a) $a_i < l$ and $b_i > l$ (Case I). The parts of the interval shaded in gray correspond to E1 or E3.



(b) $a_i \geq l$, $a_i \leq r$ and $b_i > l$ (Case I). The parts of the interval shaded in gray correspond to E2 or E4.



(c) $a_i < r$ and $b_i > l + 2\pi$ (Case II). The parts of the interval shaded in gray correspond to E1 or E5.



(d) $a_i \geq r$ and $b_i > l + 2\pi$ (Case III). The parts of the interval shaded in gray correspond to E1 or E7.

Figure 7.16: Illustration for the MIP. Shown is the interval $[0, 4\pi)$, and the left and right conflict range for two labels ℓ_i and ℓ_j (shaded in light red). The active range $[a_i, b_i)$ of the label ℓ_i is indicated above the interval $[0, 4\pi)$. The parts of the interval shaded in gray are the parts of the interval where the label ℓ_j can be active without causing a conflict.

7.7.2.2 *Model*

We have already stated at the beginning of the preceding section that in this model we want to directly compute the active range endpoints. Therefore, we maintain for each label ℓ_i two continuous variables a_i and b_i that represent the beginning and the end of its active range, i. e., the active range of ℓ_i is (a_i, b_i). Also, recall that we have "unrolled" the interval $[0, 2\pi)$ to $[0, 4\pi)$ to obtain proper intervals as active ranges. Hence, we add two continuous variables a_i and b_i for each label ℓ_i to our model along with the following constraints:

$$0 \leq a_i \leq 2\pi \tag{7.6}$$

$$0 \leq b_i \leq 4\pi \tag{7.7}$$

$$a_i \leq b_i \tag{7.8}$$

$$b_i - a_i \leq 2\pi \tag{7.9}$$

We can already state the objective function as this is straightforward. To solve MaxTotal we want to maximize $\sum_{1 \leq i \leq n}(b_i - a_i)$.

To change the result of our MIP formulation such that it finds the optimal solution to MaxMin we add a continuous variable z to our model, and for every label ℓ_i we add the following constraint.

$$z \leq (b_i - a_i) \tag{7.10}$$

We use the MIP then to solve MaxMin optimally by setting the objective function to z and maximizing it, i. e., maximize z.

Since the constraints (7.6)–(7.9) already ensure that we have a single contiguous active range per label, to guarantee that the solution is valid, we only need to ensure that no two labels are active when they are in conflict. In Section 7.7.2.1 we have given a detailed description how to determine when the intersection of an active range of a label with a conflict range is non-empty; see cases I)–III). In the following we discuss how to handle a *single* soft conflict between two labels ℓ_i and ℓ_j with active ranges (a_i, b_i) and (a_j, b_j), respectively. The conflict range of the conflict we consider is the interval $[l, r]$. We need to repeat the steps described in the following for every conflict range between any two labels of the instance. Each time we need to add *new* variables and *new* constraints.

Next we describe the solution to handle soft conflicts. Afterwards we proceed by describing how to handle hard conflicts (which is significantly easier).

SOFT CONFLICTS – HIGH-LEVEL IDEA. The high-level idea of the approach to handle soft conflicts, is that we first add a single binary variable for each of the cases I)–III). Further, we add constraints such that should one of the cases apply, the corresponding binary variable, which we refer to as the *indicator variable*, has value 1. We also add all constraints E1–E7 to our model, but slightly modified such that they can be "activated" or "deactivated" (this is done with a standard technique used in the formulation of MIPs. We explain how this works later on.). Should one of the indicator variables have value 1, we ensure that at least one of the constraints among E1–E7 is "activated" that is necessary to ensure the solution remains valid; see Table 7.1.

As next step we describe how to add constraints such that the indicator variables actually indicate whether a case applies. Then, we describe how to add the constraints E1–E7 such that they can be "activated" or "deactivated" when necessary.

SOFT CONFLICTS – DETERMINE IF A CASE APPLIES. We now want add three indicator variables I_1, I_2, I_3 to our model that indicate whether a case applies. More specifically, we want that should case I) apply $I_1 = 1$, should case II) apply $I_2 = 1$, and should case III) apply $I_3 = 1$. We observe that for a case to be true exactly two conditions must be met. The idea now is to add a binary variable for each of these conditions, and add constraints that ensure that the binary variable has value 1 if its corresponding condition is met. For example, consider case I) for which $a_i < r$ and $b_i > l$ must be true. We add for $a_i < r$ the binary variable X_1 and the constraint $X_1 > (r - a_i)/(4\pi)$ to our model. It is easy to see that $(r - a_i) > 0$ if and only if $a_i < r$ and in this case $X_1 > 0$. Since X_1 is a binary variable and the value of $(r - a_i)/(4\pi)$ is at most 1 the value of X_1 must be set to 1. Similarly, we add the binary variable X_2 and the constraint $X_2 \geq (l - b_i)/(4\pi)$. Finally, we add the constraint $I_1 \geq X_1 + X_2 - 1$ to the list of constrains. Now, should case I) apply, both X_1 and X_2 have value 1 and thus also $I_1 = 1$. We note, however, that it if $I_1 = 1$ it does not necessarily mean that case I) applies. At the end of this section, after we have introduced the remaining constraints and variables, we discuss this and argue why this poses no problem for finding the optimal solution.

We proceed in a similar fashion with the other constraints. Since some of the constraints in the cases I)–III) are identical we need to add only the following six variables $X_1, \dots, X_3, I_1, \dots, I_3$ to our model along with the following constraints:

$$X_1 \geq (r - a_i)/(4\pi) \tag{7.11}$$

$$X_2 \geq (b_i - l)/(4\pi) \tag{7.12}$$

$$X_3 \geq (b_i - l - 2\pi)/(4\pi) \tag{7.13}$$

$$I_1 \geq X_1 + X_2 - 1 \tag{7.14}$$

$$I_2 \geq X_1 + X_3 - 1 \tag{7.15}$$

$$I_3 \geq (1 - X_1) + X_3 - 1 \tag{7.16}$$

SOFT CONFLICTS – ADDING THE CONSTRAINTS E1–E7. We are now ready to introduce the remaining constraints which help us to handle soft conflicts. For this we can use the constraints E1–E7 which we described earlier. However, we need to ensure that the constraints can be "activated" or "deactivated" when necessary. This must be done based on the values of the indicator variables I_1, I_2, I_3. Activating or deactivating a constraint can be done by using a large constant, usually referred to as M in the literature, in combination of a binary variable. This is a well-known trick used for modelling MIPS. For our purposes we set $M = 5\pi$.

To model all constraints for case I), we begin by adding six binary variables $A_1, \dots, A_6$, one for each of the constraints E1–E6 to our model. The idea now is that a constraint Ei ($1 \leq i \leq 6$) is activated when its corresponding binary variable A_i has value 1; otherwise it has no effect. We can model with the variables A_i the logical OR, by adding the following constraint to our model:

$$A_1 + A_2 + A_3 + A_4 + A_5 + A_6 \geq I_1 \tag{7.17}$$

This ensure that if case I) applies, and hence $I_1 = 1$, at least one of the variables A_i ($i \leq i \leq 6$) has value 1, which in turn means that A_i's corresponding constraint is activated. We now add the following constraints that correspond to the resolving constraints E1–E6.

$$b_j \leq l + (1 - A_1)M \tag{7.18}$$

$$b_j \leq a_i + (1 - A_2)M \tag{7.19}$$

$$a_j \geq b_i - (1 - A_3)M \quad \text{and} \quad b_j \leq l + 2\pi + (1 - A_3)M \tag{7.20}$$

$$a_j \geq b_i - (1 - A_4)M \quad \text{and} \quad b_j \leq a_i + 2\pi + (1 - A_4)M \tag{7.21}$$

$$a_j \geq r - (1 - A_5)M \quad \text{and} \quad b_j \leq l + 2\pi + (1 - A_5)M \tag{7.22}$$

$$a_j \geq r - (1 - A_6)M \quad \text{and} \quad b_j \leq a_i + 2\pi + (1 - A_6)M \tag{7.23}$$

Note that the constraints 7.18–7.23 correspond to the constraints E1–E6 with an additional term which guarantees that the constraint has an effect on the variables only when the binary variable it contains has been set to 1. Otherwise, the large constant M negates the effect of the constraint.

Modeling the constraints for the other two cases (cases II and III) follows the same principle. For the second case we require only two binary variables, and for the third, we require three. This step is straightforward and we omit it here. We have now added all variables and constraints necessary to handle soft conflicts between labels.

HARD CONFLICTS. It remains to add constraints that ensure that no label is active during a hard conflict. For a label ℓ_i with a hard conflict with a conflict range $[l, r]$ the label's active range must not be active during $[l, r]$. This means that either both a_i and b_i are to the left of l or they are both in between r and $l + 2\pi$. So, for each hard conflict $[l, r]$ of a label ℓ_i we add a binary variable H and the following constraints to our model.

$$b_i \leq l + (1 - H)M \tag{7.24}$$

$$a_i \geq r - HM \quad \text{and} \quad b_i \leq l + 2\pi + HM \tag{7.25}$$

CORRECTNESS. Here we discuss the correctness of our approach. It is easy to verify that the active range of a label ℓ_i has a non-active intersection with a given conflict range $[l, r]$ if one of the cases I)–III) applies. Further, the possible resolutions (E1–E7) to each case are the only possible resolutions.

While it is easy to see then, that applying the constraints corresponding to E1–E7 works as intended for our approach, the indicator variables might pose some problems. More

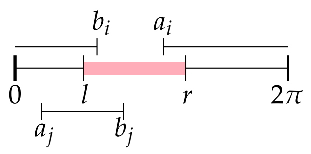


Figure 7.17: Example to show that case II) is necessary.

specifically, an indicator variable might be set to 1 even if the corresponding case does not apply. However, this does not invalidate solution found the MIP nor allows this to find better solutions since setting an indicator variable to 1 only "activates" constraints which restrict the active range of another label.

We note one possible problem with constraint (7.16). We use $(1 - X_1)$ to indicate whether $a_i \geq r$. Unfortunately, we can only be sure that if $a_i < r$ then $X_1 = 1$. Otherwise, for $a_i \geq r$ either X_1 or $1 - X_1$ has value 1. However, consider both constraints (7.15) and (7.16) (i. e., $I_2 \geq X_1 + X_3 - 1$ and $I_3 \geq (1 - X_1) + X_3 - 1$). Should $b_i > l + 2\pi$ (which is a necessary condition for case II) or case III) to apply), then $X_3 = 1$. However, in this case either I_2 or I_3 must be set to 1 (since either $X_1 = 1$ or $1 - X_1 = 1$). Further, to resolve case II) we can use only either E1 or E5, while to resolve case III) we can additionally to E1 and E5, we can use E7. This means that should $a_i \geq r$ setting $X_1 = 0$ (i. e., $(1 - X_1) = 1$) can only have beneficiary influence on the solution obtained by the MIP. Further, even setting $X_1 = 1$ should $a_i \geq r$ hold does *not* produce invalid solutions.

As we have stated in the beginning, we assume here that *all* active ranges are open intervals. Note that this does *not* change the value of the optimal solution. It has, however, unfortunate side effects. For example, consider two labels ℓ_i and ℓ_j with conflict ranges (a_i, b_i) and (a_j, b_j), where $b_i = a_j$ and where the labels are in conflict at rotation angle b_i. Since both active ranges are open intervals, none of the two labels is active at b_i. However, we can detect those cases in a post-processing step and simply change some of the active ranges to half-closed/closed intervals.

Finally, when we introduced the three different cases in Section 7.7.2.1 we pointed towards an obvious flaw in our approach. Namely, that case II) is a special case of case I) and seems to be, because of that, unnecessary. This is also reflected in constraints (7.14) and (7.15). If $I_2 = 1$ it directly implies that $I_1 = 1$ (since $X_3 \geq X_2$). It is tempting to just remove case II) and argue that this computes the same solutions. However, this might (and will) lead to incorrect solutions. The issue here is that we have not "normal" intervals but circular ones. First, note that, should case II) apply and be resolved (i. e., by constraint E1 or E5) case I) is automatically resolved to. Hence, this cannot lead to incorrect solutions. On the other hand, suppose we would remove case II) entirely. Further, imagine an instance where $a_i < r$ and $b_i > l + 2\pi$ (i. e., case I) applies). We then can resolve this conflict for example by E2 which requires $b_j \leq a_i$ and we explained it as "ℓ_i's active range ends before ℓ_i's active range begins". However, $b_j \leq a_i$ is not sufficient to avoid conflicts, since the intervals may overlap; see Figure 7.17 for an example.

RESULT. As a last step we need to retranslate the intervals that span over 2π to active ranges valid for our model; see description in Section 7.2. This can be done straightforwardly. We can conclude that the given MIP formulation obtains the optimal solution to either MaxTotal or MaxMin (depending whether or not the modification for MaxMin has been applied).

Since we have now given a full description of our MIP we can analyze the number of constraints and variables necessary for this model. It is easy to see that we require per label $O(1)$ variables and constraints, as well as $O(1)$ variables and constraints per conflict in an instance. We summarize this in the following corollary.

Corollary 7.4 *The MIP formulation solves* MAXTOTAL/MAXMIN *optimally and requires $O(cn)$ variables and constraints, where n is the number of labels, and c is the maximum number of conflicts per label in the input instance.*

The obvious advantage of this approach compared to the one presented in Section 7.7.1 is that the number of constraints and variables for a single label does not depend on the total number of conflict events, but rather only on the number of conflicts it has with other labels during a full 360° rotation. Further, this approach allows us to solve both MAXTOTAL and MAXMIN. However, it is also more complicated and relies heavily on using the concept of activating/deactivating constraints when necessary with a large constant M. This usually has a significant negative effect on the time necessary to solve the MIP. Moreover, while it is trivial to extend the ILP formulation from Section 7.7.1 such that it can find the solution in any kR model, this, while certainly possible, is not at all easy to achieve with the formulation presented in this section.

7.8 EXPERIMENTAL EVALUATION

In this section we present the experimental evaluation of different labeling strategies based on the consistency models and algorithms introduced in Sections 7.2 and 7.5– 7.7. We implemented our algorithms in C++ and compiled with GCC 4.7.1 using optimization level `-O3`. As ILP solver we used Gurobi 5.6[1]. The running time experiments were performed on a single core of an AMD Opteron 2218 processor running Linux 2.6.34.10. The machine is clocked at 2.6 GHz, has 16 GiB of RAM and 2×1 MiB of L2 cache. Before we discuss our results we introduce the benchmark instances.

7.8.1 *Benchmark Instances*

Since our labeling problem is immediately motivated by dynamic mapping applications, we focus on gathering real-world data for the evaluation. As data source we used the publicly available data provided by the OpenStreetMap project[2]. We extracted the latitudes, longitudes, and names of *all* cities with a population of at least 50 000 for six countries (France, Germany, Italy, Japan, United Kingdom, and the United States of America) and created maps at three different scales.

To obtain a valid labeling instance several additional steps are necessary. First, the width and height of each label need to be chosen. Second, we need to map latitude and longitude to the two-dimensional plane. Third, recall that the input is a statically labeled map, and hence we need to compute such a static input labeling. For the first issue we used the same font that is used in Google Maps, i. e., `Roboto Thin`[3]. The dimensions of each label were obtained by rendering the label's corresponding city name in `Roboto Thin` with font size 13, computing its bounding box, and adding a small additional buffer. For obtaining two-dimensional coordinates from the latitude and longitude of each point, we use the popular Mercator projection. For the map scales we again wanted to be close to Google Maps. Hence, we used the Mercator projection (where we approximate the ellipsoid with a sphere of radius $r = 6371$km) for three different scales (65 pixel $\widehat{=}$ 20km, 50km, 100km) for each country. For simplicity we refer to the scale of 65 pixel $\widehat{=}$ 20km only by 20km (and likewise for the remaining scales). The last remaining step was to compute a valid input labeling. For this we used the 4P fixed-position model [86] and solved a simple ILP model to obtain a weighted maximum independent set in the label conflict graph, in which any two conflicting label positions are linked by an edge and weights are proportional to

1 Gurobi is a commercial ILP solver `www.gurobi.com`
2 OpenStreetMap `www.osm.org`
3 Roboto Font www.google.com/fonts/specimen/Roboto

Table 7.2: Country instances. Number of labels in each benchmark instance, the number of labels in the largest connected component (lcc) in the conflict graph, and the number of connected components (cc) in the conflict graph.

	countries					
	FR	DE	GB	IT	JP	US
scales	#labels (#labels in lcc / #cc)					
20km	86 (12/51)	52 (20/26)	99 (73/19)	131 (28/48)	99 (12/34)	403 (26/203)
50km	80 (39/9)	43 (39/4)	68 (66/2)	111 (87/5)	80 (69/7)	359 (88/89)
100km	69 (69/1)	33 (33/1)	37 (37/1)	68 (68/1)	49 (44/3)	288 (213/16)

Table 7.3: City instances. Number of labels for each of our city instances, the number of labels in the largest connected component in the conflict graph, and the number of connected components in the conflict graph.

	cities			
	Berlin	London	New York	Paris
scales	#labels (#labels in lcc / #cc)			
20m	2744 (13/2205)	1661 (40/1044)	629 (9/445)	2367 (39/1560)
50m	2739 (77/1329)	1620 (175/569)	621 (50/279)	2325 (244/703)
100m	2628 (416/634)	1457 (371/289)	602 (88/143)	2141 (952/230)

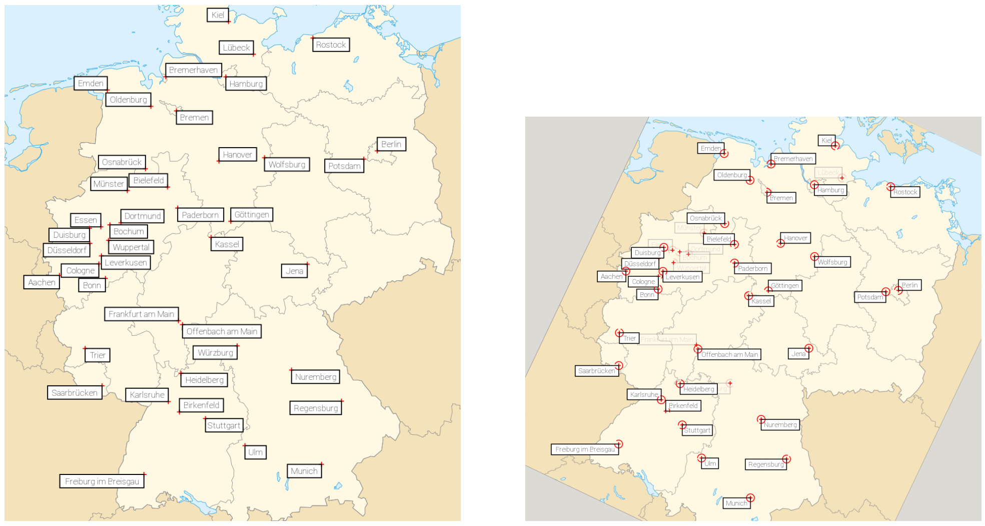
the population. Table 7.2 shows the characteristics of our benchmark data, which can be downloaded from our website[4].

For the first problem we followed the example set forth by Google Maps and chose as font Google's own Roboto[5] Font with font size set to 13. For the dimension we then chose the bounding box of the cities name rendered in `Roboto Thin` with a small additional buffer. For obtaining two-dimensional coordinates from the latitude and longitude we use the Mercator projection [172] (similar to the projection used by Google Maps). For the scales used we again wanted to be close to Google Maps depiction. Hence, we used the Mercator projection for three different scales (65 pixel $\widehat{=}$ 20km, 50km, 100km) for each country. For simplicity we refer to the scale of 65 pixel $\widehat{=}$ 20km only by 20km (and likewise for the remaining scales). The last remaining step was to compute from this data a valid labeling input instance. We took all anchor points and allowed for each label four positions. In each position the a labels corner coincides with its anchor point (this model is known as the 4P fixed-position model [86]). Then, we computed a maximum weighted independent set of the intersection graph of all those labels. As weight we chose the population of the city it corresponds to. This problem is well-known to be NP-hard [86] but it can be solved optimally with a straightforward ILP formulation.

In Table 7.2 we report the number of labels for each of our input instances. In the following we refer to the input instances by a combination of their ISO code names and the scale. For example, the instance of France with a scale of 20km is referred to by FR20.

To obtain instances that are much larger than the country instances described in Table 7.3 we chose Berlin, London, New York, and Paris and extracted the names, as well as longitude

4 Benchmark data set `i11www.iti.kit.edu/projects/dynamiclabeling/`
5 Roboto Font `http://www.google.com/fonts/specimen/Roboto`



(a) Original instance. Red crosses are the anchor points.

(b) Rotated instance with optimal solution. The red circle around the anchor points indicate for which angles the label is active.

Figure 7.18: Germany with scale 50km as displayed by our GUI.
Background picture is in public domain. Retrieved from http://commons.wikimedia.org/wiki/File:Germany_localisation_map_2008.svg

and latitude, of all restaurants in each city from OpenStreetMap. To obtain valid input data we conducted the same steps as described for the country instances with `Roboto Thin` but with font size 8 and the three scales 65pixel $\widehat{=}$ 20m, 50m, 100m. We report the number of labels for these instances in Table 7.3.

Unfortunately, the city instances are too large to obtain optimal solutions in either the 1R soft-conflict or 1R hard-conflict model in a reasonable time frame. We therefore do *not* include these instances in the evaluation of the labeling models. However, we use these instances later when evaluating the performance of the proposed algorithmic approaches.

7.8.2 *Evaluation the Model*

In this section we evaluate the different consistency models introduced in Section 7.2. The models differ by the admissible number of active ranges per label and the handling of hard conflicts. We begin by analyzing the effect of limiting the number of active ranges and consider the five models 0/1, 1R, 2R, and ∞R, all taking hard conflicts into account. As discussed in Section 7.2, the 0/1-model is flicker-free but expected to have a low total activity, especially in dense instances. On the other hand, the ∞R-model achieves the maximum possible total activity in any valid labeling, but is likely to produce a large number of flickering effects. Still, it serves as an upper bound on the total activities of the other models. The two most important quality criteria in our evaluation are (i) the total activity of the solution, and (ii) the average length of the active ranges. Unfortunately, it proved too time consuming for our ILP to solve the city instances in a reasonable time frame. Thus, we chose to restrict ourselves in this analysis to the country instances described in Table 7.2.

In Table 7.4(a) we report the total activity of the optimal solution for the tested models relative to the solution in the ∞R-model. The results of the instances are aggregated by scale. For a more detailed description we refer the reader to Table 7.5. We observe that the total activity of the 0/1-model drops to less than 55% compared to the optimal solution in

Table 7.4: Average total activity of the optimal solutions for the country instances with respect to the maximum possible objective value with standard deviation in brackets. Instances grouped by scale. Additionally we report the average interval length normalized to one full rotation, and for the ∞R-model the average number of intervals per label.

model	0/1	1R		2R		3R		∞R	
scale	tot. act.	tot. act.	∼len	tot. act.	∼len	tot. act.	∼len	∼len	∼intervals
20km	54.04%	94.56%	0.76	99.36%	0.56	99.92%	0.47	0.08	19.13
50km	22.42%	87.79%	0.58	97.69%	0.35	99.54%	0.26	0.01	79.23
100km	6.19%	81.01%	0.44	95.83%	0.27	99.24%	0.19	0.01	128.40

(a) Hard-conflict model.

model	0/1	1R		2R		3R		∞R	
scale	tot. act.	tot. act.	∼len	tot. act.	∼len	tot. act.	∼len	∼len	∼intervals
20km	69.18%	98.17%	0.83	99.79%	0.64	99.98%	0.59	0.54	1.80
50km	49.47%	94.80%	0.69	99.27%	0.44	99.86%	0.37	0.27	5.09
100km	42.41%	91.67%	0.58	98.57%	0.34	99.76%	0.24	0.07	9.55

(b) Soft-conflict model.

the ∞R-model even for the least dense instance at scale 20km and to only 6% for a scale of 100km. Hence this model is of very little interest in practice.

We see a strong increase in the average total activity values already for the 1R-model compared to the optimal solution in the 0/1-model. For the large-scale instance 20km 1R reaches almost 95% of the ∞R-model, which has more than 19 times the number of flickering effects and active ranges of average length shorter by a factor of 1/9. For map scales of 50km and 100km, the total activities drop to 88% and 81%, respectively, but at the same time the number of flickering effects and the average active range lengths in the ∞R model are extremely poor. Thus the 1R-model achieves generally a very good labeling quality by using only one active range per label.

Finally, we take a look at the middle ground between the 1R- and the ∞R-models. It turns out that total activity of the 2R-model is off from the ∞R-model by less than 1% at scale 20km and less than 5% at scale 100km, but this increase in activity over the 1R model comes at the cost of producing twice as many flickering effects and decreasing the average active range length by 30–40%. If we allow three active ranges per label, the total activity increases to more than 99% of the upper bound in the ∞R-model at all three scales, while having significantly fewer flickering effects and longer average active ranges. The activity gain by considering the kR-model for $k > 3$ is negligible and the disadvantage of increasing the number of flickering effects dominates.

When we conduct the same analysis for the 0/1, 1R, 2R, and ∞R model in the soft-conflict model, similar trends can be observed. We report in Table 7.4(b), analogous to Table 7.4(a), the total activity of the optimal solution for the tested models relative to the solution in the ∞R-model. We observe that the trend in both tables is similar. However, the 0/1-model performs in general significantly better than in the hard-conflict model, but the results are still not suitable from a practical point of view. Already the 1R model produces results

Table 7.5: Total activity normalized by 2π of the optimal solutions for each of the country instances. We also report the average length of each interval (also normalized by 2π), and the average number of intervals, where it makes sense to do so. The instances are named by a combination of country ISO code and scale.

model	0/1	1R		2R		∞R	
scale	tot. act.	tot. act.	∼len	tot. act.	∼len	∼len	∼intervals
FR20	53	72.37	0.84	73.45	0.67	0.19	4.54
FR50	15	53.59	0.67	58.32	0.42	0.03	23.02
FR100	2	28.74	0.42	35.40	0.26	<0.01	129.29
DE20	29	41.33	0.79	43.00	0.60	0.06	15.23
DE50	11	28.47	0.66	31.01	0.38	0.01	56.32
DE100	1	15.27	0.46	17.74	0.28	0.01	71.12
GB20	28	67.30	0.68	74.05	0.43	0.01	62.25
GB50	2	29.45	0.43	34.79	0.26	<0.01	139.38
GB100	0	14.02	0.38	16.84	0.23	<0.01	82.83
IT20	50	102.59	0.78	107.10	0.54	0.06	14.54
IT50	8	57.04	0.51	65.09	0.30	0.01	116.26
IT100	1	27.73	0.41	32.85	0.24	<0.01	124.75
JP20	32	68.26	0.69	72.67	0.49	0.08	9.24
JP50	9	42.80	0.53	47.51	0.31	<0.01	94.70
JP100	2	22.81	0.46	26.36	0.28	0.01	72.14
US20	195	322.18	0.80	336.82	0.61	0.09	8.97
US50	100	241.66	0.67	260.46	0.45	0.02	45.66
US100	29	148.60	0.52	173.35	0.32	<0.01	290.28

which are close to the maximum possible solution in the ∞R-model, while the difference in the optimal solutions in the 2R and 3R-model to the ∞R-model are negligible.

We conclude that the 1R-model achieves the best compromise between total activity value and low flickering, at least for maps at larger scales with lower feature density. For dense maps the 2R- or even the 3R-models yield near-optimal activity values while still keeping the flickering relatively low. Going beyond three active ranges per label only creates more flickering but does not provide noticeable additional value.

It remains to investigate the impact of hard conflicts. For this we apply the 1R-model and compare the variant where all conflicts are treated equally (soft-conflict model) with the variant where hard conflicts are disallowed (hard-conflict model). We consider for each map scale the average relative increase in activity value of the soft-conflict model over the stricter hard-conflict model. For 20km instances the increase is on average 8.51% (standard deviation 2.91), for the intermediate scale 50km it is on average 19.25% (standard deviation 7.86), and for the small-scale map 100km the increase reaches on average 31.9% (standard deviation 4.72). We report these results in detail in Table 7.6(a), and give the summary in Table 7.6(b). They (the results) indicate that, unsurprisingly, the soft-conflict model improves the total activity at all scales, and in particular for dense configurations of point features, where labels usually have several hard conflicts with nearby features. As

Table 7.6: Comparison of the total activity of the optimal solution for the country instances in the 1R hard-conflict and the 1R soft-conflict model. The total activity is normalized by 2π.

	hard-conflict	soft-conflict	
instance	tot. act.	tot. act.	Δ (+%)
FR20	72.37	75.71	4.61
FR50	53.59	60.39	12.67
FR100	28.74	39.14	36.14
DE20	41.33	44.51	7.69
DE50	28.47	31.99	12.34
DE100	15.27	19.33	26.62
GB20	67.30	75.95	12.85
GB50	29.45	39.49	34.10
GB100	14.02	19.19	36.81
IT20	102.59	109.29	6.53
IT50	57.04	70.33	23.30
IT100	27.73	38.06	37.27
JP20	68.26	76.35	11.85
JP50	42.80	51.48	20.28
JP100	22.81	29.60	29.74
US20	322.18	346.47	7.53
US50	241.66	272.51	12.76
US100	148.60	185.42	24.77

(a) Results per instance.

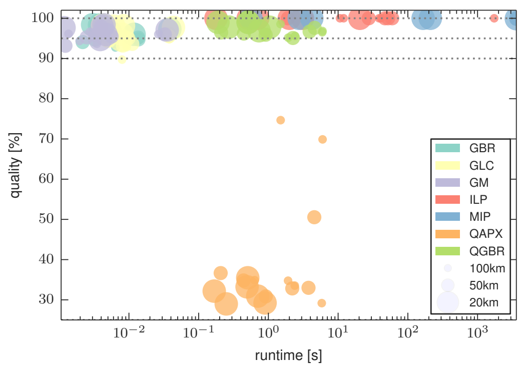
scale	tot. act. (std)
20km	+8.51% (2.91)
50km	+19.25% (7.86)
100km	+31.90% (4.72)

(b) Results aggregated by scale.

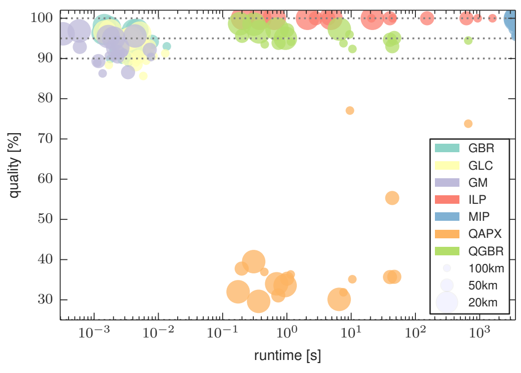
discussed before, this improvement comes at the cost of temporarily occluding unlabeled but possibly important points. It is an interesting open usability question to determine user preferences for the two models and the actual effect of temporary point occlusions on the readability of dynamic maps, but such a user study is out of scope of this evaluation and left as an interesting direction for future work.

7.8.3 *Evaluation of the Algorithms*

In this section we evaluate the quality (total activity) and running time of the 1/4-approximation algorithm (Section 7.5.1) and the three greedy heuristics GreedyMax, GreedyLowCost, and GreedyBestRatio (Section 7.6), which we abbreviate as QAPX, GM, GLC, and GBR, respectively. Additionally, we include the ILP, and the MIP (Sections 7.7.1 and 7.7.2) as exact method in the evaluation. The ILP is also applied to optimally solve the independent subinstances in the grid cells created by QAPX. In our implementation we heuristically improve the running time of the ILP/MIP by partitioning the conflict graph of the labels into its connected components and solving each connected component individually; see Table 7.2 for the number of labels in the largest connected component and the number of



(a) 1R hard-conflict model



(b) 1R soft-conflict model

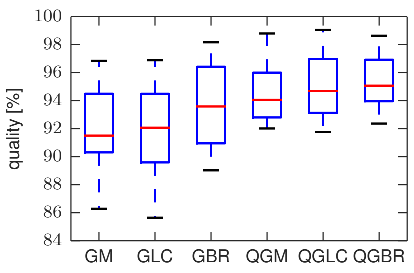
Figure 7.19: Plot of the runtime (log scale) and quality results of the algorithm evaluation for all city instances in the 1R hard-conflict and the 1R soft-conflict model.

connected components in the conflict graph of each instance. For the ILP/MIP we set a time limit of 1 hour and restrict the ILP solver to a single thread. The same restrictions are applied to the ILP when solving the small subinstances in algorithm QAPX. By the design of the algorithm, a solution obtained by QAPX will consist of many labels that have no active range, although they could be assigned one (all labels that are discarded to obtain independent cells have active range set to length 0). To overcome this drawback, we propose a combination of QAPX with the greedy algorithms. More specifically, we apply one of our greedy algorithms to each of the four solutions computed by the 1/4-approximation and determine among the four resulting solutions the best one. In the following we refer to the combination of the 1/4-approximation with a greedy algorithm by adding a Q in front of the greedy algorithm's name (e. g., QGLC). We report the results of the 1R-soft-conflict model and the 1R-hard-conflict model, since both the hard- and the soft-conflict model seem to be sensible models.

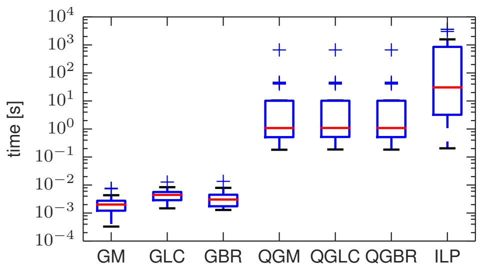
EVALUATION. We give a general overview of the performance of all evaluated algorithms as a scatter plot in Figure 7.19. In this scatter plot each disk represents the result of an algorithm (indicated by color) applied to a single country instance. The size of the circle indicates the scale of the instance (the smaller the circle, the smaller the scale). We omitted the algorithms QGM and QGLC in this plot to increase readability, because the difference in running time and quality of the solutions between the three algorithms QGM, QGLC, and QGBR is negligible and would create overplotting.

We observe that for both the hard-conflict and the soft-conflict model the performance characteristics are very similar. Tthe performance of the greedy algorithms is very good with respect to running time as well as quality of the solutions. As expected, the total activity of QAPX is always better than 25%, but generally much worse than for the remaining algorithms. It never gets close to the solutions produced by the greedy algorithms while being considerably slower. However, combining QAPX with a greedy algorithm achieves better solutions than greedy algorithms and QAPX alone, while the increase in running time over QAPX is negligible. Finally, we observe that the ILP solves the tested instances in a reasonable time frame.

A brief comparison of the two optimal algorithmic approaches (i. e., the ILP and MIP described in Sections 7.7.1 and 7.7.2) clearly demonstrates that the ILP approach is far superior to the MIP based approach on the evaluated instances. To obtain an optimal solution the ILP required on average 114s in the 1R-hard-conflict model, and 758s in the soft conflict model, while the MIP required on average 2777s in the 1R hard-conflict model,



(a) Quality of the solutions as a percentage of the (optimal) ILP solution.



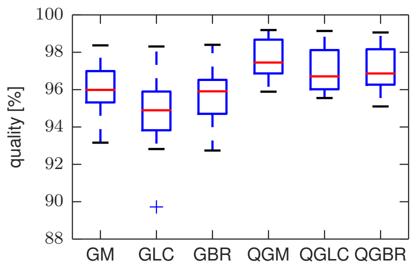
(b) Runtime (log scale) of the algorithms.

Figure 7.20: Performance of the greedy algorithms and QAPX with greedy postprocessing for the city instances in the 1R soft-conflict model.

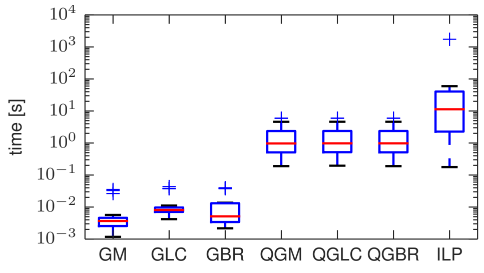
and for the solver did not finish in the set time limit of one hour for any of the instances in the 1R soft-conflict model. However, one redeeming quality of the MIP approach is that it was nearly always able to find the optimal solution, it merely lacked the proof that the solution found was indeed optimal. The average quality of each solution was 99.96% of the optimal solution. We note that since the number of constraints for the MIP is significantly lower than for the ILP there was no instance in the set of instances we evaluated where the MIP was faster than the ILP.

We now turn to a more detailed analysis of the two most promising approaches (i) using the greedy algorithms, and (ii) combining QAPX with the greedy algorithms. We begin by discussing the results in the 1R soft-conflict model. For a detailed depiction of the performance of the algorithms with respect to the quality of the solution see the diagrams in Figure 7.20. We observe that in the 1R soft-conflict model among the three greedy algorithms GBR performs best with respect to quality with an average of 93.7%, but the difference to the other greedy algorithms is small. Even the greedy algorithm GM with the lowest total activity produces solution with an average of 91.8% of the optimal solution. Each of the combinations of QAPX with subsequent execution of a greedy algorithm outperforms each of the greedy algorithms alone in terms of quality. However, since the solutions produced by the greedy algorithms are already very close to the optimal solution, we observe only a slight increase in total activity for QGM, QGLC, and QGBR over the greedy algorithms. The difference between both approaches becomes much more visible when considering the running time. While the average running time for the three greedy algorithms is between 2.5ms and 3.9ms, the average running time for the 1/4-approximation algorithms is roughly 46s. However, we note that this large difference is mostly caused by one instance, which required over 664s to find the solution. The median running time for the enriched 1/4-approximation algorithms is about 1.08s.

The results for the 1R hard-conflict model are similar to those in the soft-conflict model; see the diagram in Figure 7.21. Among the three greedy heuristics the simplest algorithm (i. e., GM) performs best with respect to quality with an average of 96%, but as a whole the greedy algorithms perform similarly well. Even the worst greedy algorithm GLC produces solutions of high quality (average is 95%). Again, each of the combinations of the 1/4-approximation with subsequent execution of a greedy algorithm outperforms each of the greedy algorithms alone. However, since the solutions produced by the greedy algorithms are so close to the optimal solution (even closer than in the soft conflict model), we observe only a slight increase in the quality of the solution for QGM, QGLC, and QGBR. Again, the difference is much more noticeable when considering the running time. While the average running time for the three greedy algorithms is between 7ms and 13ms, the average running time for the 1/4-approximation is roughly 1.8s with a median of 0.96s. We note an



(a) Quality of the solutions as a percentage of the (optimal) ILP solution.



(b) Runtime (log scale) of the algorithms.

Figure 7.21: Performance of the greedy algorithms and QAPX with greedy postprocessing for the city instances in the 1R hard-conflict model.

increase in the running time of the greedy algorithms compared to the soft-conflict model. However, all three algorithms are still very fast and suitable for real-time applications. For the combination of the 1/4-approximation and the greedy algorithms, with the exception of one instance, the running time of the 1/4-approximation in the 1R hard-conflict model is very similar to the running time in the 1R soft-conflict model. This might be surprising since the running time of the 1/4-approximation is dominated by obtaining optimal solutions for all of the subinstances by the ILP, which performs better in the hard-conflict model. However, the subinstances required for the 1/4-approximation are very small and we conjecture that this is the reason why the performance increase is not that noticeable.

CITY INSTANCES. As stated in the beginning of this section, we were not able to obtain optimal solutions of the city instances in either the 1R soft-conflict or 1R hard-conflict model. We therefore cannot compare the quality of the obtained solutions by the greedy algorithms, and by the combination of 1/4-approximation and greedy algorithms with the optimal solution. Instead, we chose to compare the results of the algorithms with the optimal solution in the ∞R-model, which serves as an upper bound on the optimal solution in the 1R-, and 2R-model; see Table 7.7.

We observe that the results for all algorithms in all models are very close to the maximal achievable total activity in the ∞R model. Even in the most dense instance in the 1R hard-conflict model, the worst greedy algorithm achieves still on average about 86% of the maximal achievable total activity with a running time of about .8s. The quality of the greedy algorithms is slightly better in the 1R soft-conflict model and the running time is much smaller (around 30-40ms). The combination of 1/4-approximation and greedy heuristic does perform slightly better, but the difference is minuscule. However, the running time differs drastically. While in the city instances before, the average running time was about 1s in both the 1R hard-conflict and the 1R soft-conflict model, here, the running time is in the most dense instances on average about one minute in the soft-conflict model, and in the hard-conflict model about 20 minutes on average. For the 2R-model we note that the running time of the 1/4-approximation based algorithm decreases significantly in both models (but most noticeable in the soft-conflict model), but the algorithms still require several seconds to find a solution. The running time of the greedy algorithms is about .04s in the soft-conflict model and about 0.8s in the hard-conflict model but this seems to be independent from the kR model used.

The results indicate that the greedy algorithms in the 1R model produce solutions that are very close to the maximum possible total activity while the algorithms require only few milliseconds in running time. This strengthens our conclusion that the 1R model in combination with any of the three greedy algorithms is the best strategy for labeling rotating

Table 7.7: Evaluation of the three greedy algorithms, and the combination of 1/4-approximation with the greedy algorithms for the city instances; see Table 7.3. We compare the total activity of the solutions with the optimal solution in the ∞R-model. The average performance of the greedy algorithms given under "G", while the performance of the QAPX and subsequent greedy algorithms is given under "Q".

(a) 1R hard-conflict model

	algorithms			
	G		Q	
scales	time	tot. act.	time	tot. act.
20m	0.78s	97.39%	4.29s	97.58%
50m	0.84s	92.53%	16.93s	92.92%
100m	0.83s	86.61%	54.55s	87.17%

(b) 1R soft-conflict model

	algorithms			
	G		Q	
scales	time	tot. act.	time	tot. act.
20m	0.03s	98.58%	1.30s	98.70%
50m	0.04s	95.52%	238.51s	95.82%
100m	0.05s	91.39%	1124.27s	92.02%

(c) 2R hard-conflict model

	algorithms			
	G		Q	
scales	time	tot. act.	time	tot. act.
20m	0.72s	99.67%	3.88s	99.77%
50m	0.79s	98.65%	5.90s	98.86%
100m	0.80s	97.04%	11.02s	97.31%

(d) 2R soft-conflict model

	algorithms			
	G		Q	
scales	time	tot. act.	time	tot. act.
20m	0.03s	99.69%	1.02s	99.76%
50m	0.04s	98.34%	14.65s	98.73%
100m	0.06s	96.17%	29.84s	96.87%

maps. Whether to use the soft-conflict model or the hard-conflict model is a design choice that should depend on the requirements of the actual application.

In order to give a final recommendation for an algorithm, it is necessary to make a choice on the time-quality trade-off that is acceptable in a particular situation. If running time is not the primary concern, e.g., for offline applications with high computing power available, we can recommend the ILP, which ran reasonably fast in our experiments, at least for the smaller country instances. On the other hand, if computing power is limited, instances are large, or real-time labeling is necessary, e.g., on a mobile device, all three greedy heuristics can be recommended as the methods of choice; a slight advantage of GBR was observed in our experiments. All three algorithms run very fast (a few milliseconds) and empirically produce high activities of more than 90% of the optimum solution. If one wants to invest some extra running time, the combination of QAPX with a greedy algorithm may be of interest as it produces slightly better solutions than the stand-alone greedy algorithms and is much faster than the ILP.

7.9 CONCLUSION

We have introduced a new model for consistent labeling of rotating maps and proved NP-hardness of the active range maximization problem. We could, however, show that there is an EPTAS for the MAXTOTAL problem that works for rectangular labels with arbitrary anchor points and bounded height ratio, width ratio, and aspect ratio. An interesting open question and an important challenge in practice is to combine map rotation with zooming and panning and study the arising algorithmic labeling problems. Besides the

EPTAS we presented three simple greedy heuristics and two different (mixed) integer linear programming models to takle the presented problems and generate high-quality labelings in the 1R hard-conflict model.

In the second part of this chapter, we evaluated different strategies for labeling dynamic maps that allow continuous rotation. A labeling strategy consists of a consistency model and a labeling algorithm. We compared the optimal solutions in the 0/1-, 1R-, and 2R-models with the optimal solution in the ∞R-model on several real-world instances. We concluded that the most promising results are obtained in the 1R-, and 2R-models, since the optimal solutions in these models reduce flickering significantly, while still being close to the maximum objective value. However, since one of our main goals is to reduce flickering and the 1R-model performs even in dense instances quite well we chose to investigate the 1R-model further. The comparison of the 1R hard-conflict and the 1R soft-conflict model indicated that the 1R soft-conflict model allows solutions to have much higher objective value than in the 1R hard-conflict model. This concluded one part of our analysis of labeling strategies. For the next part we chose to investigate the 1R soft-conflict model further.

Finally, we evaluated the proposed algorithmic approaches. All three greedy heuristics performed very well in both running time and total activity. The solutions achieve on average a total activity of well above 90% of the optimal solution, while requiring only few ms in running time. The 1/4-approximation itself performs much worse than any of the greedy heuristics. However, the combination of 1/4-approximation and greedy algorithms performs with respect to quality slightly better than the greedy heuristics alone, but the running time can be several seconds. In conclusion, we argue that the 1R soft-conflict model in combination with any of the three greedy algorithms is, in most cases, the best labeling strategy for labeling rotating maps.

Although we have shown NP-hardness for MaxTotal and MaxMin for the 1R hard-conflict model the proof presented in Section 7.4 does not work for the 1R soft-conflict model, nor does it work in the kR hard-conflict model for any $k \geq 2$. As an open problem we propose to investigate the computational complexity of MaxTotal and MaxMin in these models. Further, we did not propose any algorithms (besides the MIP) that produce solutions for MaxMin. The 1/4-approximation can obviously not be used and the greedy algorithms do not take special care to maximize the minimum active range length. Developing algorithms for MaxMin may be a new interesting research topic. Another, interesting modification to our approach is to reduce unwanted flickering by requiring that if a label is active it stays active for a certain fraction of the full rotation (this ensures that a label is not only briefly visible during a rotation) and if a label gets invisible at some point, it stays invisible for a certain fraction of a full rotation.

8

LABELING OF ZOOMING MAPS WITH SLIDING LABELS

We study a dynamic labeling problem for points on a line that is closely related to labeling of zoomable maps. Typically, labels have a constant size on screen, which means that, as the scale of the map decreases during zooming, the labels grow relatively to the set of points, and conflicts may occur due to overlapping labels. Our algorithmic problem is a combined dynamic selection and placement problem in a sliding-label model: (i) select for each label ℓ a contiguous *active range* of map scales at which ℓ is displayed, and (ii) place each label at an appropriate position relative to its anchor point by sliding it along the point. The *active range optimization* (ARO) problem is to select active ranges and slider positions so that no two labels intersect at any scale and the sum of the lengths of active ranges is maximized. We present a dynamic programming algorithm to solve the discrete k-position ARO problem optimally and an FPTAS for the continuous sliding ARO problem.

This chapter is based on joint work with Martin Nöllenburg, and Ignaz Rutter. Preliminary results have been published in [105].

8.1 INTRODUCTION

With the increasing practical importance of dynamic maps that allow continuous operations like zooming, panning, or rotations, dynamic labeling of map features becomes a critical aspect of the visual quality of a map. Examples of dynamic maps range from maps on small-screen mobile devices to professional desktop GIS applications. The map dynamics add new dimensions to label placement, which result in challenging geometric optimization problems that are quite different from static labeling problems. Changes in dynamic maps due to continuous map movements need to be smoothly animated in order to preserve a coherent context and minimize the user's cognitive load for re-orientation [154]. This requirement is also known as "frame coherency" [4] or "temporal continuity" [18]. Hence we cannot simply solve the arising labeling problems independently for each intermediate map view during the animation; rather we need to solve the labeling problem globally such that the animations of all possible trajectories using the given set of navigation operations satisfy the quality constraints.

In order to avoid distraction and irritation of the user a dynamic map should—according to Been et al. [9]—adhere to the following quality constraints or *consistency desiderata* for dynamic map labeling: During monotone map movement labels should neither "jump" (non-continuously change position or size) nor "pop" (vanish when zooming in or appear when zooming out); moreover, the labeling should be a function of the selected map viewport and not depend on the navigation history. In this chapter we are only interested in dynamic labelings that are consistent in that sense. Of course each static map view in a dynamic map also needs to satisfy the quality standards for static maps [123], i.e., all labels—usually modeled as rectangles—are pairwise disjoint, each label is close to its anchor point, and, globally over all possible map views, the number of visible labels is maximum.

Been et al. [10] presented a first extensive study of algorithms for dynamic map labeling in several different models for one- and two-dimensional input point sets. However, they focused on the *dynamic label selection* problem, i.e., which set of labels to select at which scale, and assumed that for each label a single, fixed position relative to the anchor point is given in the input. They left *dynamic label placement* as an open problem, i.e., the problem where to place each label relative to its anchor point. One model for label placement is the *k-position* or *fixed-position* model, where each label can be placed at a position from a set of k (usually 4 or 8) possible positions [2, 181, 86]. Another more general model is the *slider* model, where the finite-position assumption is dropped and each label can take any position such that the anchor point coincides with a point on the label boundary [136, 174]. In this chapter we present labeling algorithms in a dynamic scenario that allows continuous zooming for the visualization of a one-dimensional input point set. To the best of our knowledge our algorithms are the first to combine the dynamic label selection problem with label placement in both the fixed-position and the slider model, thus answering (partially) an open question of Been et al. [10]. Although this problem seems purely artificial zooming of time lines can be seen as an application for this kind of problem; see Figure 6.8.

RELATED WORK. Most previous algorithmic research on automated label placement deals with *static* fixed-position or slider models for point, line, or area features. The problem of maximizing the number of selected labels is NP-hard even for the simplest labeling models, whereas there are efficient algorithms for the decision problem that asks whether all points can be labeled in some of the simpler models (see, e.g., the discussion by Klau and Mutzel [131] or the comprehensive map labeling bibliography [186]). Approximation results [136, 2], heuristics [182], and exact approaches [131] are known for many variants of the static label number maximization problem.

More recently, *dynamic* map labeling has emerged as a new research topic that gives rise to many unsolved algorithmic problems. Petzold et al. [156] used a preprocessing step to generate a reactive conflict graph that represents possible label overlaps for maps of all scales. For any fixed scale and map region, their method computes a conflict-free labeling in the slider model using heuristics. Poon and Shin [159] described algorithms for labeling one- and two-dimensional point sets that precompute a hierarchical data structure storing solutions for a number of different scales; this allows them to answer adaptive zooming queries efficiently. Mote [147] presented another fast heuristic method for dynamic conflict resolution in label placement that does not require preprocessing and assumes a 4-position model. The consistency desiderata of Been et al. [9] for dynamic labeling, however, are not satisfied by any of these three methods. Been et al. [10] showed NP-hardness of the label number maximization problem in the consistent labeling model and presented several approximation algorithms for labeling two-dimensional point sets and an exact algorithm for one-dimensional point sets. They focused on dynamic label selection, i.e., assumed a 1-position model for label placement. Nöllenburg et al. [151] recently studied a dynamic version of the alternative boundary labeling model allowing continuous zooming and panning, where labels are placed at the sides of the map and connected to their points by leaders. Algorithms and complexity results for dynamic label selection in fixed-scale rotating maps that satisfy similar consistency desiderata were presented by Gemsa et al. [104].

CONTRIBUTION. In this chapter we present algorithms for labeling a set of points on a line with labels of arbitrary, non-uniform length in a dynamic scenario that supports continuous zooming of the points' visualization. Unlike previous efforts [10] we consider label placement in a k-position and slider model: we must select both an interval of scales at which each label is selected (dynamic selection problem) and an admissible label position for each label relative to its anchor point (dynamic placement problem). We require that the label position remains the same for all scales. In Section 8.2 we introduce a model for dynamic point labeling with sliding labels in the framework of Been et al. [9]. Section 8.3 presents a dynamic programming algorithm for dynamically labeling points in the k-position model. Our main contribution is the fully polynomial-time approximation scheme (FPTAS) described in Section 8.4 for the more general sliding model. Since the time complexity of our proposed algorithms might be too large for large values of n, we briefly describe approximation algorithms and how they can be extended to the two-dimensional case in Section 8.5. Finally, we conclude in Section 8.6 with several remaining open questions that arise from our results.

8.2 PRELIMINARIES

In this section we describe our model for dynamic labeling in the general framework of Been et al. [9, 10].

base line
anchor point

MODEL. Let $P = \{p_1, \ldots, p_n\}$ be a set of points on the x-axis (also called the *base line*) together with a set $\mathcal{L} = \{\ell_1, \ldots, \ell_n\}$ of labels. The point p_i is called the *anchor point* of the label ℓ_i. Each label ℓ_i is a rectangle of (target) width w_i modeling the bounding box of the text describing the point p_i. Since we focus on labeling a one-dimensional point set, we can think of each label ℓ_i as actually being a line segment of width w_i. During zooming of the points' visualization we wish to keep the label size constant on screen, which means that if we scale the map by a factor of $1/s$ we need to increase the label size by a factor of s in order to maintain its width on screen constant. This is the *label size invariance property* of Been et al. [9]. So the width of ℓ_i on the base line required for a map of scale $1/s$ is given by the linear function $w_i(s) = w_i s$.

The label proximity constraint in map labeling says that each label must be close to its anchor point [123], i.e., we require for each label that the anchor point coincides with a point of the label. We consider sliding labels and define the shift position $t_i \in [0, 1]$ of a label ℓ_i as the fraction of ℓ_i that is to the right of p_i. For $t_i = 0$ the label is in its leftmost position, and for $t_i = 1$ it is in its rightmost position. In the fixed-position model only a finite subset of positions from $[0, 1]$ is allowed. In this chapter we consider *invariant point placements* [9], i.e., once a shift position t is selected for a label ℓ, ℓ maintains that position relative to its anchor point. This immediately prevents the labels from jumping. Figure 8.1 shows a set of five points with labels zoomed to four different scales. Note that as the scale decreases, the points move closer together and some labels must be removed to avoid conflicting labels.

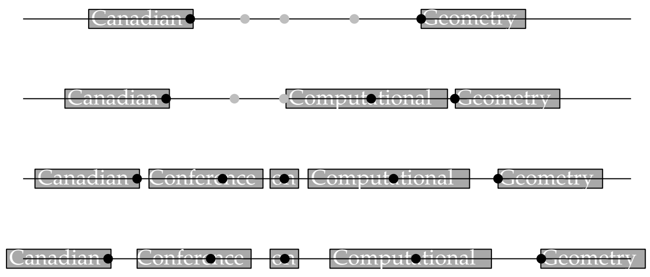


Figure 8.1: Five (partially) labeled points on a line zoomed from smaller (top) to larger scales (bottom).

scale factor
active

Following Been et al. [10, 9] we define an extended two-dimensional coordinate system defined by the x-axis, which models the positions of the points P, and the y-axis, which models the inverse s of the scale $1/s$. We denote s as the *scale factor* that is used to enlarge the labels before the whole base line (including the labels) is scaled down by the target scale $1/s$ to produce the actual visualization. We say a label is *active* at scale factor s if it is selected as being visible at s; otherwise it is *inactive*. The (static) *placement* of an active label ℓ with target width w and anchor point p at scale factor s is determined by a shift position $t \in [0, 1]$, i.e., the label is represented by the interval $[p - (1 - t)ws, p + tws]$. A

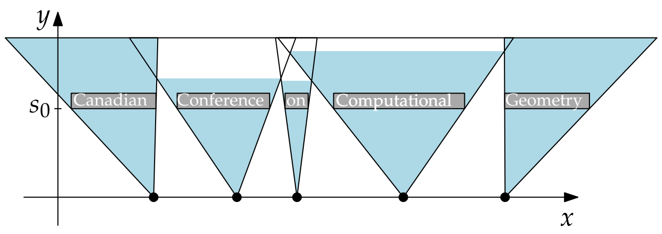


Figure 8.2: Triangular truncated extrusions (shaded blue) induced by the example of Figure 8.1.

dynamic placement of ℓ is a placement of ℓ for each scale factor s at which ℓ is active. Since we consider invariant point placements, the shift position is the same for all scales. If we extrude the growing label segment with its constant shift position t along the y-axis from $y = 0$ to some maximum scale factor $s_{\max}$ we obtain a triangle whose apex is placed at the point p and whose top side is parallel to the x-axis, see Figure 8.2. We call this triangle the *extrusion* E of ℓ. The shift position t determines the slant of E, but for a label ℓ of width w the width of E at any fixed scale factor s is ws independent of t. Let the *trace* $\mathrm{tr}_s(E)$ of E at scale factor s be the intersection of E with the horizontal line $y = s$. By definition $\mathrm{tr}_s(E)$ corresponds to the placement of ℓ at s if ℓ is active at s.

extrusion

trace

If the extrusions E and E' of two labels intersect at some scale factor s this means that the two labels ℓ and ℓ' overlap at scale $1/s$. A standard requirement in point labeling, however, is that all labels must be pairwise disjoint [123]. Accordingly, at most one of ℓ or ℓ' can be active at scale factor s. Since one of the desiderata for consistent dynamic map labeling is that labels do not 'pop' during monotonous zooming in order to avoid flickering effects [9] we require that labels never vanish when zooming in and never appear when zooming out. This lets us define the *active range* of a label ℓ_i as an interval of scale factors $[1, a_i)$ for which ℓ_i is active. This active range implies that when zooming in the label ℓ_i appears exactly once at scale factor a_i and then remains active, or, conversely, when zooming out it disappears exactly once at scale factor a_i and remains inactive. The *truncated extrusion* T_i is the restriction of the extrusion E_i of ℓ_i to its active range $[1, a_i)$, see Figure 8.2 for an example. Now a consistent dynamic labeling for the points P corresponds to an assignment of a scale-independent shift position t_i and an active range $[1, a_i)$ for each label ℓ_i such that the truncated extrusions $\mathcal{T} = \{T_1, \ldots, T_n\}$ are pairwise disjoint. Hence we need to solve both a dynamic selection problem and a dynamic placement problem according to Been et al. [9]. Informally speaking, we can adjust the slant and the height of the truncated extrusions as long as they do not intersect each other.

active range

truncated extrusion

OBJECTIVE. A common objective in point labeling is to maximize the number of labeled points, and accordingly our goal is to maximize the *total active range length*, which is defined as the sum $H = \sum_{i=1}^{n} a_i$ of all active range lengths. Maximizing H corresponds to displaying a maximum number of labels integrated over all scale factors $s \in [1, s_{\max}]$. This problem is known as the active range optimization problem (ARO) [10]. We consider two one-dimensional variants of ARO: In the discrete *k-position 1d ARO problem* the set of admissible shift positions is restricted to a subset $\mathcal{S}_i \subset [0, 1]$ of cardinality $|\mathcal{S}_i| \leq k$. In the general *sliding 1d ARO problem* any shift position in $[0, 1]$ is admissible.

total active range length

8.3 A DYNAMIC PROGRAM FOR k-POSITION 1D ARO

In this section we give a dynamic program for computing an optimal solution for the k-position version of the 1d ARO problem. For ease of notation we define two dummy points p_0 and p_{n+1}, where $p_0 = \min\{p_i - s_{\max} w_i \mid 1 \leq i \leq n\}$ and $p_{n+1} = \max\{p_i +$

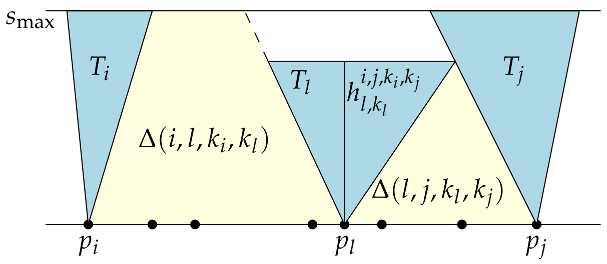


Figure 8.3: The subinstance induced by $\Delta(i,j,k_i,k_j)$ is split into two smaller independent subinstances by T_l.

$s_{\max} w_i \mid 1 \leq i \leq n\}$. The only shift position of ℓ_0 is $\mathcal{S}_0 = \{0\}$ and the only shift position of ℓ_{n+1} is $\mathcal{S}_{n+1} = \{1\}$. Both labels have width 1. It is easy to see that in any optimal solution the height of T_0 and T_{n+1} must be $s_{\max}$ since none of the extrusions T_i can intersect T_0 or T_{n+1}.

For a pair of points p_i and p_j with $i < j$ and shift positions $k_i \in \mathcal{S}_i$ and $k_j \in \mathcal{S}_j$ we define the free space $\Delta(i,j,k_i,k_j)$ as the polygon bounded by the line $s = 0$, the supporting line of the right edge of T_i in shift position k_i, the supporting line of the left edge of T_j in shift position k_j, and, if the two supporting lines of T_i and T_j do not intersect below $s_{\max}$, the line $s = s_{\max}$. See Figure 8.3 for an example. Let $\mathcal{A}[i,j,k_i,k_j]$ be the maximum total active range height for the points $p_{i+1},\dots,p_{j-1}$, where all truncated extrusions $T_{i+1},\dots T_{j-1}$ are contained in $\Delta(i,j,k_i,k_j)$.

We observe that the tallest truncated extrusion T_l $(i < l < j)$ in any optimal solution of the subinstance I induced by $\Delta(i,j,k_i,k_j)$ must touch the left, right, or top boundary of $\Delta(i,j,k_i,k_j)$, otherwise we could improve the total active range height. We use T_l in order to split I into two smaller independent subinstances I' and I'' induced by $\Delta(i,l,k_i,k_l)$ and $\Delta(l,j,k_l,k_j)$; see Figure 8.3. For each $l = i+1,\dots,j-1$ let h_{l,k_l}^{i,j,k_i,k_j} denote the height at which T_l at shift position $k_l \in \mathcal{S}_l$ first hits a non-bottom edge of $\Delta(i,j,k_i,k_j)$. We initialize $\mathcal{A}[i,i+1,\cdot,\cdot] = 0$ for all $i = 0,\dots,n$ and then recursively define $\mathcal{A}[i,j,k_i,k_j] = \max\{\mathcal{A}[i,l,k_i,k_l] + h_{l,k_l}^{i,j,k_i,k_j} + \mathcal{A}[l,j,k_l,k_j] \mid i < l < j \text{ and } k_l \in \mathcal{S}_l\}$. By definition of $\mathcal{A}$ the solution to the ARO problem is $\mathcal{A}[0,n+1,0,1]$. We can compute the value $\mathcal{A}[0,n+1,0,1]$ by dynamic programming in $O(n^3k^3)$ time: each of the $O(n^2k^2)$ values in $\mathcal{A}$ is defined as the maximum of a set of $O(nk)$ values, each of which can be computed by two table look-ups and two $O(1)$-time line intersection queries.

The correctness of the above dynamic program follows by induction on the number of points in a subinstance. Clearly for an empty subinstance $\Delta(i,i+1,k_i,k_{i+1})$ the maximum total active range height $\mathcal{A}[i,i+1,k_i,k_{i+1}]$ is 0. Let's consider a subinstance induced by $\Delta(i,j,k_i,k_j)$, where $j - i = r$ and assume by induction that the values in $\mathcal{A}$ are correct for all subinstances $\Delta(i',j',k_{i'},k_{j'})$, where $j' - i' < r$. Let $\mathcal{B}$ be an optimal active range assignment of the labels $\ell_{i+1},\dots,\ell_{j-1}$ within the free space $\Delta(i,j,k_i,k_j)$ and let $H(\mathcal{B})$ be its value. Let further T_l be a tallest truncated extrusion with shift position k_l in $\mathcal{B}$. Obviously T_l must have height h_{l,k_l}^{i,j,k_i,k_j} if $\mathcal{B}$ is optimal. Since our algorithm explicitly considers all labels and all shift positions as candidates for the tallest truncated extrusion, it also considers T_l and its shift position k_l, which splits the given instance into two independent subinstances with free spaces $\Delta(i,l,k_i,k_l)$ and $\Delta(l,j,k_l,k_j)$. Since $l - i < r$ and $j - l < r$ we know that $\mathcal{A}[i,j,k_i,k_j] \geq \mathcal{A}[i,l,k_i,k_l] + h_{l,k_l}^{i,j,k_i,k_j} + \mathcal{A}[l,j,k_l,k_j] = H(\mathcal{B})$.

Since the free space $\Delta(0,n+1,0,1)$ is chosen such that none of the truncated extrusions $T_1,\dots,T_n$ can touch T_0 or T_{n+1}, $\mathcal{A}(0,n+1,0,1)$ indeed contains the value of an optimal solution to the k-position ARO problem. We can easily augment the algorithm to keep track of the pair (l, k_l) that achieved the maximum value in order to reconstruct the solution by backtracking from $\mathcal{A}(0, n+1, 0, 1)$. We summarize this result in the following theorem.

Theorem 8.1 *Given n points $P = \{p_1, \ldots, p_n\}$ on the x-axis, a label ℓ_i of base width w_i for each point p_i, and a set $\mathcal{S}_i \subset [0, 1]$ of at most k shift positions for each label ℓ_i, we can compute an optimal solution to the k-position 1d ARO problem in $O(n^3k^3)$ time and $O(n^2k^2)$ space.*

We note that this algorithm generalizes the $O(n^3)$-time algorithm of Been et al. [10] for the 1-position 1d ARO problem, where each label has only a single available shift position.

8.4 AN FPTAS FOR GENERAL 1D SLIDING ARO

In this section we present an FPTAS for approximating the optimal solution of the sliding 1d ARO problem within a factor of $(1 - \varepsilon)$. The idea of the FPTAS is based on uniformly discretizing the interval $[0, 1]$ of shift positions. Let $k > 0$ be an integer and define the set of shift positions $\mathcal{S}^k = \{i/k \mid i \in \mathbb{Z}, 0 \leq i \leq k\}$. For an instance I of the sliding 1d ARO problem consisting of a point set $P = \{p_1, \ldots, p_n\}$ and corresponding label set $\mathcal{L}$, we consider instead the $(k+1)$-position 1d ARO problem for the instance I' consisting of P, $\mathcal{L}$, and the shift position sets $\mathcal{S}_i = \mathcal{S}^k$ for $1 \leq i \leq n$. By Theorem 8.1 this instance I' can be solved in $O(n^3k^3)$ time and $O(n^2k^2)$ space using the dynamic programming algorithm of Section 8.3. In the following theorem we show that this approach gives an FPTAS for the original sliding 1d ARO problem.

Theorem 8.2 *Given n points $P = \{p_1, \ldots, p_n\}$ on the x-axis and a corresponding label set $\mathcal{L} = \{\ell_1, \ldots, \ell_n\}$, where label ℓ_i has base width w_i, we can compute a $(1 - \varepsilon)$-approximate solution to the sliding 1d ARO problem in $O(n^3(1/\varepsilon)^3)$ time and $O(n^2(1/\varepsilon)^2)$ space.*

Proof. We need to show that for a suitably chosen parameter $k = k(\varepsilon)$ the optimal solution for the $(k+1)$-position 1d ARO instance I' as defined above is actually a $(1 - \varepsilon)$ approximate solution for the sliding 1d ARO instance I. Let us assume that we know an optimal solution $A^\star$ for I, i.e., a shift position $t_i \in [0, 1]$ and an active range $[1, a_i) \subseteq [1, s_{\max}]$ for each label ℓ_i. Since $A^\star$ is optimal, each truncated extrusion T_i has either height $a_i = s_{\max}$ or touches the left or right supporting line of another, taller, truncated extrusion T_j. In the latter case a_i is the smallest scale factor, where the extrusions E_i and E_j intersect.

For proving the approximation factor we derive a discretized solution A' from $A^\star$, where each shift position $t'_i \in \mathcal{S}^k$ and the active ranges are shortened to $[1, a'_i) \subseteq [1, a_i)$ as to satisfy the label disjointness property. For every shift position t_i in $A^\star$ we define the new shift position t'_i in A' as follows

$$t'_i = \begin{cases} \lfloor kt_i \rfloor / k & \text{if } t_i < 1/2 \\ \lceil kt_i \rceil / k & \text{if } t_i \geq 1/2. \end{cases}$$

In other words, we tilt T_i towards its "heavier" side until it reaches a shift position in the set $\mathcal{S}^k$. Due to the tilting the truncated extrusions are no longer necessarily disjoint and we need to shorten the active ranges for some labels. Figure 8.4 shows an example.

Let T'_i and T'_j be two tilted truncated extrusions that intersect in their interior. Without loss of generality let T'_i be the smaller one such that, before the tilting, its top right corner was touching the left edge of T'_j as in Figure 8.4(a). We first consider the case that T'_i and T'_j are tilted towards each other. Then the right edge of T'_i, the left edge of T'_j, and the horizontal line $y = a_i$ define a triangle D as in Figures 8.4(b) and 8.5. We decrease the active range of label ℓ_i to $[0, a'_i)$, where $a'_i = a_i - h$ for the height h of D. Obviously the truncated extrusions T'_i and T'_j no longer intersect in their interior for the new active range $[0, a'_i)$.

Next, we bound the height h of D. Let α be the angle between the right edge of T'_i and the x-axis. The same angle α is found at the top right corner of D. From Figure 8.4(b) we obtain that $\tan(\alpha) = a_i/(w_i t'_i a_i) = 1/(w_i t'_i)$ and from Figure 8.5 that $\tan(\alpha) = h/x$, where x

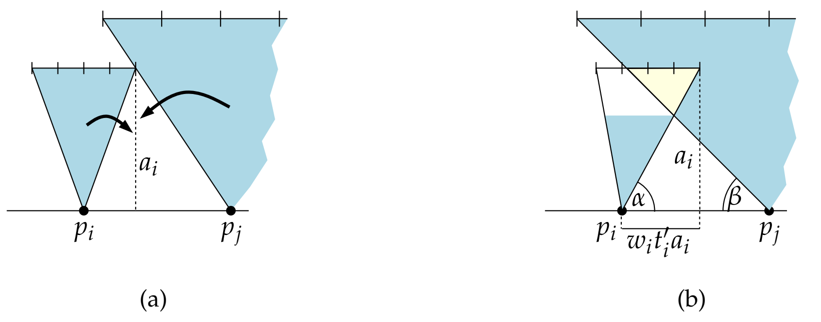


Figure 8.4: Discretizing the shift positions of two labels for $k = 4$.

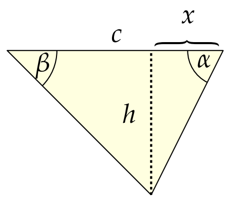


Figure 8.5: Intersection triangle D.

is distance between the base point of the height h on the top side c and the top right corner. Since T'_i is tilted to the right and T'_j to the left we have $t'_i \geq 1/2$ and $(1 - t'_j) \geq 1/2$. By definition of the new shift positions t'_i and t'_j we know that the length of the top side c of D is at most $(w_i a_i + w_j a_i)/k$. This is because at scale factor s the tilt moves each truncated extrusion with base width w horizontally by at most a $1/k$ fraction of its width ws at s. With $x \leq c$ this yields

$$h = \frac{x}{w_i t'_i} \leq \frac{c}{w_i/2} \leq \frac{2a_i}{k} \frac{w_i + w_j}{w_i}. \tag{8.1}$$

Similar reasoning for the angle β in Figures 8.4(b) and 8.5 yields $\tan(\beta) = a_i/(w_j(1-t'_j)a_i) = h/(c-x)$ and subsequently $h \leq 2a_i/k \cdot (w_i + w_j)/w_j$. Again without loss of generality we assume that $w_i \geq w_j$ and obtain $\min\{(w_i + w_j)/w_i, (w_i + w_j)/w_j\} = (w_i + w_j)/w_i \leq 2$. So we can finally bound the height of D by $h \leq 4a_i/k$.

We still need to consider the case that both truncated extrusions are tilted in the same direction, say to the left (the case that both are tilted to the right is symmetric). A conflict can still occur if T'_j is tilted further to the left than T'_i. The triangle D is defined as before, but now we know that the length of the side c is at most $w_j a_i/k$. Since T'_j is tilted to the left we still have $(1 - t'_j) \geq 1/2$. Now we argue about the angle β using the same identities as before and obtain $h \leq c/(w_j(1 - t'_j)) \leq 2a_i/k$.

In the case that T'_i and T'_j are tilted away from each other obviously no conflict can occur. Furthermore, the analysis still holds for conflicts involving the top left corner of T'_i instead of the top right corner.

So each truncated extrusion T_i of height a_i is shortened by at most $4a_i/k$ due to the discretization of the shift positions, or, equivalently, $a'_i \geq (1 - 4/k)a_i$. If we set $k = 4/\varepsilon$ we arrive at $\sum_{i=1}^{n} a'_i \geq (1 - \varepsilon) \sum_{i=1}^{n} a_i$. □

8.5 APPROXIMATION ALGORITHMS

We have discussed the k-position model and given an optimal algorithm for this case which has a running time that is cubic in n, the number of labels, and k. Since this time complexity may be unsuitable for large instance we propose two approximation algorithms with better

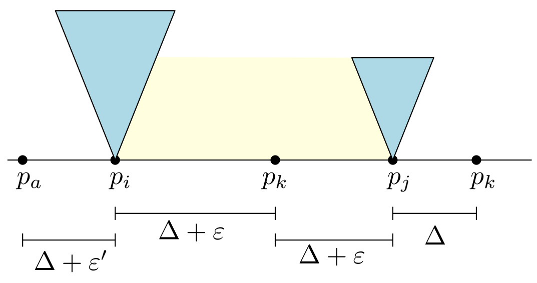


Figure 8.6: Illustration of the proof for Theorem 8.3.

time complexity. We begin with a simple approximation algorithm for the case $k = 1$ for unit-width labels where the shift position of all labels is the same.

In previous work by Been et al. [10] an approximation algorithm, called top-to-bottom fill down sweep, for the case $k = 1$ with running time $O(n^2)$ has been proposed. In the following we describe how to improve the running time to $O(n \log n)$ for the 1-position 1d ARO problem. The algorithm by Been et al. for this case is a simple greedy algorithm. It iteratively selects the label (or cone) which can be assigned the maximum active range starting at $s = 1$ among all labels. Ties are broken arbitrarily. A naive implementation yields a $O(n^2)$ time complexity. In the following we describe a modification of the algorithm with $O)(n \log n)$ time complexity.

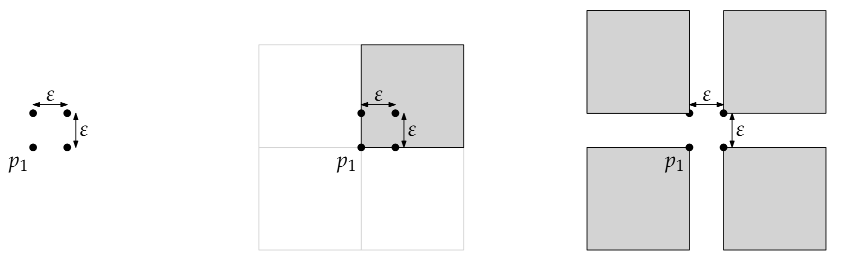
First, note that in this scenario the y-coordinates of the intersections of the extrusions of all labels is independent of the shift position chosen for the labels, since changing the shift position can also be seen as a simple a horizontal shearing of the extrusions. Hence, in the following we assume that the shift position $t = 0.5$ has been chosen.

We now describe the algorithm. It begins by setting the active range of the leftmost label to $[1, s_{\max}]$, and the active range of the rightmost label either to $[1, \min\{s_{\max}, y_{1n}\}]$, where y_{ab} is the y-coordinate of the first intersection point of the extrusion of ℓ_a with the extrusion of ℓ_b. Now, the algorithm selects then the label ℓ_i whose anchors lies closest to the midpoint between p_1 and p_n and assigns ℓ_i its maximum active range which is $[1, \min\{s_{\max}, y_{1i}, y_{in}\}]$. This separates the instance into two smaller subinstances, namely $\ell_1, \ldots, \ell_i$ and $\ell_i, \ldots, \ell_n$. In both subinstances we follow the same idea: select the label that is closest the midpoint between the anchors of the subinstance's left- and rightmost label; set the active range of this label to the minimum of $s_{\max}$, and the y-coordinates of the intersection of the labels extrusion with the truncated extrusion of the left- and rightmost label in the subinstance. The algorithm repeats until all labels have been assigned an active range.

Theorem 8.3 *There is an $O(n \log n)$-time approximation algorithm for 1-position 1d ARO for unit-width labels and with approximation ratio* 1/2.

Proof. We begin by proving the claimed time complexity. First, we observe that after initially sorting of the anchors (which requires $O(n \log n)$ time), we can set the active ranges of ℓ_1 and ℓ_n in $O(1)$. Then, for each subinstance we need to find the label whose anchor is closest to the midpoint between the anchor of the left- and rightmost label. Since the set of anchors is sorted, we can do this with binary search in $O(\log n)$ time. In each step of the algorithm we set the active range of one label, hence, we require $O(n)$ such steps, which yields a total time complexity of $O(n \log n)$.

It remains to show the correctness of the algorithm. Note that the algorithm always considers subinstances $\ell_i, \ldots, \ell_j$ that are sorted from left to right. In the following we assume that the algorithm correctly computed the step before the subinstance was created. It is easy to see that the algorithm computes a valid solution for the first three labels it considers (the leftmost and rightmost labels of the instance, as well as the label closest to the middle point between those two). Further, we assume w.l.o.g. that the active range of ℓ_i is at least as long as the active range of ℓ_j; see Figure 8.6.



(a) Instance. (b) Worst-case example. (c) Optimal solution.

Figure 8.7: Illustration showing that the approximation ratio of $1/4$ is tight.

First, it is easy to see that should the active ranges of ℓ_i and ℓ_j have active ranges that end at $s_{\max}$ choosing ℓ_m and assigning it $[1, \min\{s_{\max}, y_{im}, y_{mj}\}]$ as active range selects the label that can have the longest active range and produces no overlap with labels already assigned an active range. Further, the same argument holds should the truncated extrusion of ℓ_i intersect the truncated extrusion of ℓ_j. Hence, we assume that at least ℓ_j has an active range that is strictly shorter than $s_{\max} - 1$ (but also ℓ_i may have a shorter active range) and that the truncated extrusions of both ℓ_i and ℓ_j do not intersect. Now, we need to argue that ℓ_m's active range must be y_{mj}.

Assume that ℓ_m's active range can be $[1, y_{mz}]$ for some label ℓ_z to the right of ℓ_j without creating an invalid solution. We know that the truncated extrusion of ℓ_j must intersect the truncated extrusion of a label ℓ_k to its right (k might be z but also ℓ_z might also lie to the right of ℓ_k). Let the distance between p_j and p_k be denoted by Δ. Since ℓ_m's active range is longer than the active range of ℓ_j the distance between p_m and p_j must be $\Delta + \varepsilon$ for some $\varepsilon > 0$. Further, since the truncated extrusion of ℓ_i and ℓ_m do not intersect the distance between both their anchors must also be at least $\Delta + \varepsilon$. Now, we take a step back and consider the subinstance that was considered by the algorithm before, from which the current subinstance was created. It was either $\{\ell_a, \ldots, \ell_i, \ldots, \ell_j\}$ or $\{\ell_i, \ldots, \ell_j, \ldots, \ell_k\}$. However, in both subinstances the label ℓ_m lies closer to the midpoint between the left- and rightmost label than ℓ_i, and ℓ_j, respectively. Hence, the algorithm would not have chosen either of them which is a contradiction to the initial assumption. Applying this argument iteratively yields the proof of correctness.

Finally, since the algorithm operates in principle exactly as the algorithm proposed by Been et al. the approximation ratio follows directly from their proof.

□

Restricting ourselves to $k = 1$ is very limiting. In the next step we propose an approximation algorithm for any $k \geq 1$ for label's that have unit width (are unit squares). This approximation algorithm is a straightforward extension of the already mentioned algorithm by Been et al. in their previous publication. Recall that in the algorithm the label is selected that can have the longest active range among all remaining labels. To adapt it for the k-position case, we simply consider for each label all k different positions, when determining the label with longest active range. The correctness argument by Been et al. still applies, and the running time is $O(k^2n^2)$. We further note that this also directly applies to the two-dimensional case with the exception that the approximation ratio is $1/4$. We summarize these results in the following theorem.

Theorem 8.4 *There are approximation algorithms with time complexity $O(k^2n^2)$ for the k-position 1d ARO (for labels with unit width) and k-position 2d ARO (for labels that are unit squares) with approximation ratios $1/2$, and $1/4$ respectively.*

This approximation ratio is tight in the 4P fixed-position model, where the anchor coincides with one of the labels corners; see Figure 8.7(a) depicted is an instance consisting of four anchors which are very close together (their x-/y-coordinates differ by at most a small ε). If the algorithms selects the lower left point first and selects the extrusion to the anchor's upper right, the remaining anchors can have active ranges with length close to 0. In the optimal solution each label has an active range of $[1, s_{\max})$; see Figure 8.7(c).

8.6 DISCUSSION

In this chapter we studied an extension of the initial ARO problem, introduced by Been et al. [10], where we additionally allow to slide the labels. Our dynamic programming approach for discrete k-position 1d ARO is a generalization of their approach to solve simple 1d ARO with proportional dilation. It shows that k-position 1d ARO can be solved in polynomial time.

Based on the dynamic program for k-position 1d ARO, we further derived an FPTAS for sliding 1d ARO by suitably discretizing the set of allowed shifts for the labels. While this shows that we can approximate the optimal value arbitrarily closely in polynomial time, the complexity of sliding 1d ARO is still open. The main difficulty in devising an NP-hardness proof is that the problem becomes efficiently solvable when every label has only a polynomial number of relevant sliding positions. It thus seems difficult to encode binary decisions as label positions.

Note that in our model the shift position of each label, once selected, remains fixed for all scales. For the k-position model this is actually required in order to avoid jumping labels, whereas in a more general sliding model we leave as an open problem to determine a continuous function that defines the label position for every scale. Here we might require that this function is monotone or that its slope is bounded.

In practice it is common that some points are more important than others and hence the active ranges of their labels should be more influential in the objective function. More precisely, let $\gamma_i > 0$ be a weight for each point p_i. We can then optimize the weighted total active range length $H_\gamma = \sum_{i=1}^{n} \gamma_i a_i$ instead of H. It is easy to see that both the dynamic programming algorithm and the FPTAS remain valid for optimizing H_γ.

Another problem variant is to use a non-linear objective function, motivated by the observation that H favors active labels at large values of s, i. e., in maps with small scales $1/s$. It might be reasonable in practice to choose a logarithmic function over a linear function for measuring the active ranges. Using the objective function $H_{\log} = \sum_{i=1}^{n} \log a_i$ instead of H has the effect that doubling the scale range at which a label is active has a fixed impact on the objective function regardless of the actual scale. The dynamic programming algorithm can immediately deal with $H_{\log}$. Even the approximation scheme of Section 8.4 remains an FPTAS under the mild additional assumptions that the minimum scale factor is 1 (instead of 0), that each $a_i \geq 2$, and that $\varepsilon \leq 1/2$.

Although we have briefly touched the subject of 2d dynamic map labeling by giving an approximation algorithm for unit-square labels in the k-position model, this field is still widely unexplored. One of the main challenges there is to consistently support multiple modes of interaction (zooming, panning, rotations) using a slider model for the labels. In this sense, our results are a first step towards consistent dynamic labeling of 2d zoomable maps with sliding labels. Unfortunately, our dynamic programming based algorithms do not easily generalize to 2d point sets. In fact, it can be easily seen that k-position 2d ARO and sliding 2d ARO are both NP-hard [10]. However, the simple extensions from Section 8.5 of the approximation algorithm by Been et al. [10] to handle the k-position model in the 1d and 2d case for unit square labels. For the sliding 2d ARO problem deciding whether all labels may be active at all scales amounts to deciding whether all labels can be placed at scale $s_{\max}$. A slight modification of the NP-hardness proof for map labeling in the four-slider model [136] shows that this problem is NP-hard, even if all labels are unit squares.

9

LABELING PANORAMAS

Boundary labeling deals with placing annotations for objects in an image on the boundary of that image. This problem occurs frequently in situations where placing labels directly in the image is impossible or produces too much visual clutter. Examples are annotating maps, photos, or technical/medical illustrations. Previous algorithmic results for boundary labeling consider a single layer of labels along some or all sides of a rectangular image. If, however, the number of labels is large or the labels are too long, multiple layers of labels are needed.

In this chapter we study boundary labeling for panorama images, where n points in a rectangle R are to be annotated by disjoint unit-height rectangular labels placed above R in K different rows (or layers). Each point is connected to its label by a vertical leader that does not intersect any other label. We present polynomial time algorithms based on dynamic programming that either minimize the number of rows to place all n labels, or maximize the number (or total weight) of labels that can be placed in K rows for a given integer K. For weighted labels, the problem is shown to be (weakly) NP-hard, and we give a pseudo-polynomial algorithm to maximize the weight of the selected labels. We have implemented our algorithms; the experimental results show that solutions for realistically-sized instances are computed instantaneously. Further, we have also investigated two-sided panorama labeling, where the labels may be placed above or below the panorama image. In this model all of the above-mentioned problems are NP-hard. For solving them we propose mixed integer linear program formulations.

9.1 INTRODUCTION

Annotating features of interest in images by textual labels or icons is an essential aspect of information visualization. Depending on application and image content these labels are either placed directly next to the features within the image or, if partial occlusion of the image by labels is unacceptable or if feature density is too high, on the image boundary. In the first, so-called *internal labeling model*, which is common, for example, for placing object names in topographic maps, the association between feature and label should be clear from the spatial proximity between them. This is no longer the case in the latter *boundary labeling model* and hence features and associated labels are connected to each other using simple arcs. In this chapter we consider a new and practically important boundary labeling variant, motivated by labeling features in panorama images of, for example, skylines or mountain ranges. Such labeled illustrations are frequently used for describing landmarks and buildings at lookout points or in tourist guide books. Moreover, very wide panorama photographs of streets of houses as used in popular commercial digital road maps are often annotated by information about local businesses or other points of interest. Another application in the area of augmented reality is the GeoScope [41] system that augments camera images by textual information and thereby allows a user to explore a panorama, for example, the skyline of a city. A common aesthetic requirement in all these examples is that the labels are placed above a *horizon line*, for example, in the area taken up by the sky

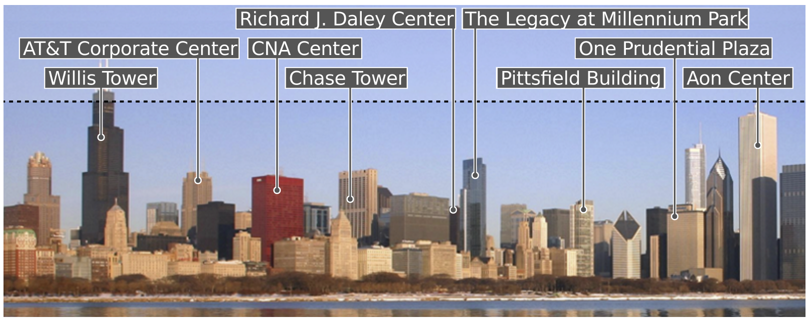


Figure 9.1: Panorama labeling for the skyline of Chicago. Photography: ©J. Crocker.

or simply above the actual image. In other cases, for example, when annotating events on a time line, a similar labeling scheme is used, where labels can be placed either above an upper horizon line or below a lower horizon line.

In this chapter we present efficient algorithms for computing optimal boundary labelings of such panorama images. Figure 9.1 shows a labeling in our model: we are given a set of n anchor points (or *sites*) in a rectangle R and for each point a variable-width but unit-height open rectangular label (think of the bounding box of the object name written as a single line of text). In order to achieve at least a horizontal proximity between points and labels, every label must be placed vertically above its associated anchor point. Each label and its associated site are connected by a vertical line segment, called *leader*. Our labeling model, in which the lower edge of a label can slide horizontally along the upper leader endpoint, can be seen as an extension of the so-called *1-slider* model [136] (in contrast to fixed-position models). The algorithmic problem is to select a subset of the labels and for each selected label compute a label position (that is, a row index and a horizontal slider position) such that no two labels overlap and no leader intersects any label other than its own. We consider two basic optimization goals: (i) minimize the number of rows required to place *all* labels, and (ii) maximize the number of labels that can be placed in K rows.

We start with a review of related work on boundary labeling and point out our main contributions.

9.1.1 *Related Work*

Algorithmic label placement problems have been studied in computational geometry for more than 20 years now [86]; a vast body of literature is collected in the map-labeling bibliography [186]. Most of the literature, however, is concerned with internal label placement as traditionally used in cartography. Boundary labeling as an alternative labeling model was first introduced as an algorithmic problem by Bekos et al. [15] and has subsequently been studied in various flavors; see also the recent survey by Kaufmann [128]. We note that in most boundary labeling problems all labels are placed in only a single row/column.

Different boundary labeling models can be classified by (a) the shape of the leaders, (b) the sides of R at which labels can be placed, and (c) further restrictions about the labels such as variable or uniform size, placement in multiple layers etc. Leaders are usually represented as polygonal lines; we argue that for readability, the leader shape should be as simple as possible, but also have a contrast high enough to be distinguishable from the background. Aiming for leader simplicity can be observed as a design goal in most manually created boundary labelings in practice and it also follows the unambiguity criterion for external labels proposed by Hartmann et al. [111]. Leaders of arbitrary orientation without bends are called *straight* or *type-s* leaders. To reduce visual clutter axis-parallel leaders are often preferred over arbitrary type-*s* leaders. The shape of an axis-aligned polygonal leader starting from the anchor point is described by a string over the alphabet $\{p, o\}$, where p and o denote, respectively, leader segments parallel and orthogonal to the side of R containing the label. If a segment is diagonal at a fixed angle (for example, 45°), we use the letter d to refer to its orientation. The letters in the string correspond to the sequence of leader segments, starting from the feature point and ending at the label port.

Bekos et al. [15] presented efficient labeling algorithms in the one-, two-, and four-sided model using type-s, type-po and type-opo leaders. Their main objective was to minimize the total leader length, but they also presented an algorithm for minimizing the number of bends in one-sided opo-labeling. Benkert et al. [19] studied algorithms for one- and two-sided po- and do-labeling with arbitrary leader-dependent cost functions (including total leader length and number of bends); the algorithms were implemented and their performance was evaluated experimentally. Nöllenburg et al. [152] presented a po-labeling algorithm for placing boundary labels that can slide along one side of the boundary so that the total leader length is minimized. Recently, Kindermann et al. [129] presented an algorithm that finds a crossing-free po-labeling where the labels are placed on two adjacent sides of R. In the same paper the authors also extend the algorithm such that it can handle three and four-sided labelings. Bekos et al. [11] presented algorithms for combinations of more general *octilinear* leaders of types do, od, and pd and labels on one, two, and four sides of R. For uniform labels the algorithms are polynomial, whereas the authors showed NP-hardness for a variant involving non-uniform labels. Recently, Huang et al. [120] considered the problem of computing boundary labelings with flexible label positions. They gave several polynomial-time algorithms for computing one- and two-sided boundary labelings with minimum leader length or minimum total number of bends.

Extensions of the basic static point-feature model include algorithms for labeling area features [13], and a dynamic one-sided po-model, in which the user can zoom and pan the map view while the label size on the boundary remains fixed [152]. Relaxing the property that each label is connected to a unique site leads to the many-to-one model. In this model NP-hardness results, approximations and heuristics for crossing minimization with type-opo and -po leaders are known [110]; Lin [140] presented an approach using duplicate labels and opo-hyperleaders to avoid crossings. Bekos et al. [16] extended the work in the many-to-one model and gave algorithms and complexity results for using po-hyperleaders, so-called *backbones*. Fink et al. [85] consider the problem of labeling focus regions on a map by using either straight lines or Bézier curves as leaders.

The only previous work using multiple layers of labels on the boundary presented $O(n^4 \log H)$-time algorithms for label size maximization in a one-sided model with two or three “stacks” of labels on a vertical side of R and type-opo leaders [12] (here H is the height of the rectangle R). In the algorithms all labels are assumed to be of uniform size and a maximum scaling factor is determined such that all labels can be placed in the available stacks. The authors further gave NP-hardness results for some two-stack variants of non-uniform labels and opo- or po-leaders.

9.1.2 *Contribution*

In this chapter we study a one-sided multi-row labeling problem with type-o leaders and variable-width labels. Note that for comparison with the results of Bekos et al. [12], the same model can be transformed into an equivalent multi-stack labeling problem with variable-height labels; since for textual annotation variable-width labels are more relevant, we describe the multi-row model. However, we argue that using a multi-stack (or multi-row) model with opo-leaders exhibits the risk of cluttered drawings. For example, multiple leaders may squeeze through a narrow gap between two labels and thus correspondences between points and labels may become unclear. In contrast, by restricting the leaders to vertical straight-line segments we try to compensate for the relatively high visual complexity of a multi-row model and to maintain the visual association.

In Section 9.2 we introduce our labeling model, in which we place all labels above the horizon, in more detail and define three optimization problems: MinRow, which aims to find a labeling with all n labels in the minimum feasible number $K^\star$ of rows, MaxLabels, which maximizes the number of labels placed in K given rows, and MaxWeight, which considers labels with integer weights, weighted by importance (with total weight w_{total}) and computes a maximum-weight subset of labels for K rows. Section 9.3 describes an $O(K^\star n^3)$-time algorithm for MinRow, an $O(Kn^3 w_{\text{total}}^2)$-time algorithm for MaxWeight, and an $O(Kn^3)$-time algorithm for MaxLabels. Additionally, we discuss the problem of generating two-sided panorama labelings, where labels may be placed above and below the respective horizons. In this context, all of the above mentioned problems are NP-hard and we cannot give efficient algorithms. Instead, we present mixed-integer linear programming formulations to solve each problem. In Section 9.4 we present extensions of the algorithms for practically interesting variations of the basic problems. We have implemented our algorithms and report results of an experimental evaluation in Section 9.5.

9.2 PANORAMA LABELING MODEL

We aim to label a set $P = \{p_1, \dots, p_n\}$ of points in the plane, where $p_i = (x_i, y_i) \in \mathbb{R}^2$ with corresponding *labels* $L = \{l_1, \dots, l_n\}$. A label is an open, axis-parallel rectangle of given width $W_i \in \mathbb{R}_0^+$ and height 1. We assume that the points in P have distinct x-coordinates and the points are ordered from left to right, that is, $x_i < x_{i+1}$ for $i = 1, \dots, n-1$. Moreover, we assume $y_i < 0$ for $i = 1, \dots, n$ meaning that p_i lies below the horizontal line $y = 0$, which we denote as the *horizon*. For our problems, the y-coordinates of the points in P are irrelevant and we can assume that all points lie on the horizontal line $y = -1$. By $P_{i,j} = \{p_i, \dots, p_j\} \subseteq P$ we denote the set of all input points between p_i and p_j.

labels

horizon

The task of labeling P comprises two subproblems: selecting the labels that are to be displayed and placing them above the horizon. More formally, we define a *(panorama) labeling* $\mathcal{L}$ to be the tuple (S, π) where $S \subseteq L$ and $\pi \colon S \to \mathbb{R} \times \mathbb{N}$ is a mapping that assigns each label l_i in S a coordinate (X_i, Y_i) where X_i is the real-valued x-coordinate of the label's right border and Y_i is an integer which allows us to say that the label is placed in *row* Y_i above the horizon.

panorama labeling

row

In order to connect each label $l_i \in S$ with its corresponding point $p_i \in P$, we draw a vertical line segment from (x_i, Y_i) to (x_i, y_i); we call this line segment the *leader* of label l_i and point p_i.

leader

We say that a labeling $\mathcal{L} = (S, \pi)$ is *feasible* if it satisfies requirements (F1)–(F3):

feasible

(F1) For every label $l_i \in S$ the leader of l_i actually connects l_i with p_i, that is, $X_i - W_i \leq x_i \leq X_i$.

(F2) For every label $l \in S$ the leader of l does not intersect any label in S other than l.

(F3) The labels in S do not intersect each other.

For a pair (i, j) of indices $1 \leq i \leq j \leq n$ and a row index k, we define an (i, j, k)*-labeling* as a feasible labeling of P_{ij} with $S = \{l_i, \dots, l_j\}$ satisfying

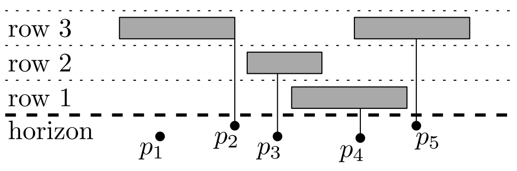
(i,j,k)-labeling

(R1) $Y_i = Y_j = k$, that is, both l_i and l_j are in row k, and

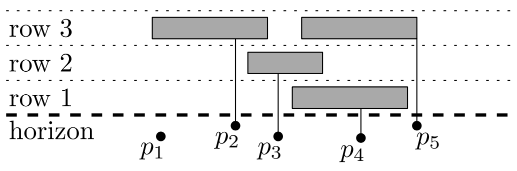
(R2) $Y_\ell \leq k$ for $\ell = i+1, \dots, j-1$, that is, the labels for all points between p_i and p_j are in row k or below.

We say that an (i, j, k)-labeling $\mathcal{L}$ is *compact* if there is no (i, j, k)-labeling where label l_j has a smaller x-coordinate than in $\mathcal{L}$; we denote the x-coordinate of l_j in a compact (i, j, k)-labeling $\mathcal{L}$ as the *x-value* $\mathcal{X}_{i,j,k}$ of $\mathcal{L}$. If no (i, j, k)-labeling exists we set $\mathcal{X}_{i,j,k} = \infty$. For an example of (non-)compact (i, j, k)-labelings see Figure 9.2.

compact (i,j,k)-labeling



(a) A non-compact $(2,5,3)$-labeling.



(b) A compact $(2,5,3)$-labeling.

Figure 9.2: Examples for a compact and a non-compact (i, j, k)-labeing

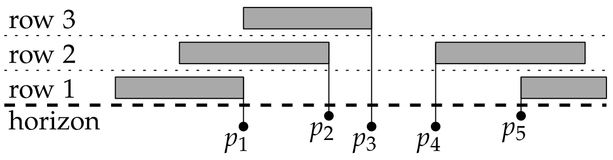


Figure 9.3: A feasible labeling for $n = 5$ labels using $\lceil n/2 \rceil = 3$ rows.

For multi-row boundary labeling there are several possible optimization criteria. Generally, we want to display as many labels as possible, decrease unused space but also ensure good readability. Note that these criteria might be conflicting, for example, putting many labels as close together as possible decreases unused space, but may also be detrimental for the readability (or more specifically, it may be harder to find the label corresponding to a specific feature point). Hence, we might want to limit the number of rows used for displaying labels. However, this may come at the price of fewer displayed labels than possible. In the following we introduce three optimization problems that we investigate in this paper.

As a first, and probably most basic, problem we want to find a solution that selects *all* labels and uses as few rows as possible.

Problem 4 (MinRow) *Given a set P of n points below the horizon and a set L of corresponding labels, find a feasible labeling with* all *labels that requires the minimum number of rows.*

A solution to this problem contains all provided information, but might be hard to read if too many features with corresponding labels are supplied. Nevertheless, there exists for every instance such a labeling.

Lemma 9.1 *For each* MinRow *instance with n points there exists a feasible labeling with $\lceil n/2 \rceil$ rows and for each $n \in \mathbb{N}$ there exists a* MinRow *instance with n points that requires $\lceil n/2 \rceil$ rows.*

Proof.

We begin by showing the first part of Lemma 9.1, that is, we show that for each MinRow instance with n points, there exists a feasible labeling with $\lceil n/2 \rceil$ rows. We label all points in P as in Figure 9.3, that is, for $i = 1, \ldots, \lceil n/2 \rceil$ we set $Y_i = i$ and $X_i = x_i$; for $i = \lceil n/2 \rceil + 1, \ldots, n$ we set $Y_i = n - i + 1$ and $X_i = x_i + W_i$. Clearly, requirement (F1) holds. Requirement (F2) holds since for $i = 1, \ldots, n-1$ and $j = i+1, \ldots, n$ label l_j lies above Y_i if $j \leq \lceil n/2 \rceil$ and to the right of x_i if $j > \lceil n/2 \rceil$. Requirement (F3) holds since each row either contains a single label or two labels l_i, l_j with $i \leq \lceil n/2 \rceil < j$. In the latter case l_i lies to the left of l_j since the right boundary of l_i is $X_i = x_i$, the left boundary of l_j is $X_j - W_j = x_j$, and $i < j$ implies $x_i < x_j$. To show that $\lceil n/2 \rceil$ rows may be required let $p_i = (i, -1)$ and $W_i = n$ for $i = 1, \ldots, n$. Since the distance between p_1 and p_n is $n - 1$ at most two labels fit in one row. □

Obviously, solutions as the one in Figure 9.3 are rather useless in practice—placing no more than two labels in a single row the required space to display all labels becomes huge. If the horizontal distances between the points are small and the labels are wide, however, we will fail to find a better solution. Therefore, a natural alternative to MinRow is to discard some labels if the available space is limited. If we are restricted to a certain number of rows, then a sensible optimization goal is to maximize the number of displayed labels.

Problem 5 (MaxLabels) *Given a set P of n points below the horizon, a corresponding set L of labels, and a positive integer K, determine a feasible labeling that displays the maximum number of labels in at most K rows.*

However, by simply maximizing the number of labels we might fail to measure the quality of a labeling appropriately. We should account for the fact that often some objects are more important than others and thus should have a higher priority to become labeled. This can be expressed with *weighted* points and yields our third optimization problem.

Problem 6 (MaxWeight) *Given a set P of n points below the horizon, a corresponding set L of labels, a positive integer K, and a positive integer weight w_i for each point $p_i \in P$, find a feasible labeling $\mathcal{L} = (S, \pi)$ that maximizes the total weight $\sum_{l_i \in S} w_i$ among all feasible labelings that use at most K rows.*

9.3 ALGORITHMS

In this section we describe our algorithms to solve the problems MinRow, MaxWeight, and MaxLabels. For MinRow and MaxLabels we give polynomial-time algorithms, while we show that MaxWeight is weakly NP-hard by a reduction from the Partition problem; we give a pseudo-polynomial time algorithm in this case.

We begin this section with a simple algorithm that decides for a given panorama labeling instance whether there exists a valid labeling in a single row, that is, whether MaxLabels has a solution with n labels for $K = 1$.

9.3.1 *Single-Row*

Before we describe our algorithm for MinRow in Section 9.3.2, we briefly discuss the following single-row label placement problem, which serves as a base case for MinRow.

Problem 7 (SingleRow) *Given a set P of n points below the horizon and a label for each point, decide whether there is a feasible labeling for P with* all *labels in a single row above the horizon.*

Bekos et al. [14] considered a similar problem, more specifically, they considered the problem of computing a boundary labeling of points on a horizontal line. However, they allow that the leaders have bends and optimize the total leader length or the number of total bends, where we require that the leaders are connected with the labels with a single vertical line.

The SingleRow problem is closely related to a single-machine job scheduling problem, where n ordered jobs $J_1 < \cdots < J_n$ with processing times z_i and release and due times r_i and d_i are to be non-preemptively scheduled in the given order such that all jobs finish before their due times. The weighted version of this problem is known as single-machine throughput maximization [7] and has been related to one-dimensional weighted point labeling problems before [160]. Note that for our problem the property that the scheduling is non-preemptive (that is, no job is interrupted) is crucial since each job corresponds to a label, and each label is displayed either completely or not at all.

SingleRow can be solved with a simple greedy algorithm, which we denote as SingleRowAlg. The algorithm processes the points in increasing x-order and places the next label l_i in the leftmost possible position such that it does not intersect the previous label, that is, $X_i = \max\{X_{i-1} + W_i, x_i\}$. If for any i we have $X_i > x_i + W_i$, then obviously no feasible single-row labeling exists (requirement (F1) violated), the algorithm reports failure and returns ∞. Otherwise it reports success and returns the position X_n of the last label. The correctness of SingleRowAlg is immediate and for sorted points it takes linear time.

Corollary 9.1 *If the input points are sorted by their x-coordinates,* SingleRow *can be solved in $O(n)$ time.*

Using SingleRowAlg we can compute the x-positions $\mathcal{X}_{i,j,1}$ of all compact $(i, j, 1)$-labelings in $O(n^2)$ time by running it n times, once for every set $P_{i,n}$, where $i = 1, \ldots, n$. If there is a feasible position for placing label l_j ($j \geq i$) in the instance $P_{i,n}$, we store its x-position X_j as the x-position $\mathcal{X}_{i,j,1}$. If there is no feasible position we set $\mathcal{X}_{i,j,1} = \infty$.

9.3.2 *Row Number Minimization*

We now show how to solve MinRow efficiently using dynamic programming. Our idea is to construct compact (i, j, k)-labelings for all $1 \leq i \leq j \leq n$ and successively increasing values of k until a feasible labeling for all points is found. Recall that by Lemma 9.1 the value of k is upper bounded by $\lceil n/2 \rceil$.

To ease notation we introduce two dummy points p_0 and p_{n+1} to the left of p_1 and to the right of p_n that do not influence the feasibility of labeling P, regardless of the position of l_0 and l_{n+1}. We set $W_0 = W_{n+1} = 0$, $x_0 = x_1 - 2W_{\max}$, and $x_{n+1} = x_n + 2W_{\max}$, where $W_{\max} = \max_{1 \leq i \leq n} W_i$ is the maximum label width.

Our algorithm MinRowAlg computes a three-dimensional table $\mathcal{T}$, where each entry $\mathcal{T}[i, j, k]$ for $0 \leq i \leq j \leq n+1$ and $1 \leq k \leq \lceil n/2 \rceil$ stores the x-position $\mathcal{X}_{i,j,k}$ of a compact (i, j, k)-labeling as well as some backtracking information in order to reconstruct the compact (i, j, k)-labeling. With these semantics it is clear that there is a solution to MinRow with k rows if and only if there is a feasible $(0, n+1, k)$-labeling, that is, $\mathcal{T}[0, n+1, k] < \infty$.

We compute $\mathcal{T}$ in a bottom-up fashion with respect to the row index k. First, we compute $\mathcal{T}[\cdot, \cdot, k]$ for $k = 1$ and if $\mathcal{T}[0, n+1, 1] = \infty$ proceed to computing $\mathcal{T}[\cdot, \cdot, k]$ for $k = 2$ and so on until eventually $\mathcal{T}[0, n+1, k] < \infty$. The entries of $\mathcal{T}[\cdot, \cdot, k]$ for $k = 1$ are computed by the algorithm SingleRowAlg described in Section 9.3.1. For $k > 1$ the entries $\mathcal{T}[i, i, k]$ for $i = 0, \ldots, n+1$ are set to $\mathcal{T}[i, i, k] = \mathcal{X}_{i,i,k} = x_i$. We use the recurrence $\mathcal{T}[i, j, k] = \min \Theta^k_{i,j}$ for all $i < j$ and $k > 1$, where $\Theta^k_{i,j}$ for $i < j$ and $k > 1$ is defined as the set

$$\Theta^k_{i,j} = \left\{ \max\{x_j, \mathcal{T}[i, \ell, k] + W_j\} \mid i \leq \ell < j,\ \mathcal{T}[i, \ell, k] \leq x_j,\ \mathcal{T}[\ell, j, k-1] < \infty \right\}. \quad (9.1)$$

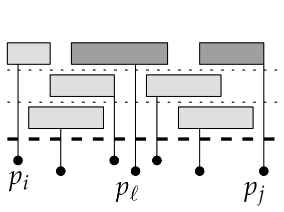
Note that $\Theta^k_{i,j}$ can be empty. For this case we define $\min \emptyset := \infty$ and obtain $\mathcal{T}[i, j, k] = \infty$. For the pseudo code of MinRowAlg see Algorithm 3.

Algorithmus 3 : MinRowAlg

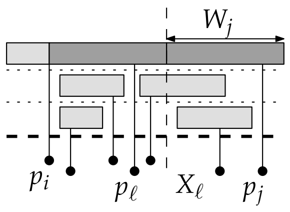
```
1 initialize $\mathcal{T}$
2 compute $\mathcal{T}[\cdot, \cdot, 1]$ using SingleRowAlg
3 for $k = 2$ to $\lceil n/2 \rceil$ do
4     for $j = 1$ to $n + 1$ do
5         for $i = 0$ to $j - 1$ do
6             $\mathcal{T}[i, j, k] = \min \Theta^k_{i,j}$
7         $\mathcal{T}[j, j, k] = x_j$
8     if $\mathcal{T}[0, n+1, k] < \infty$ then break;
9 reconstruct solution from $\mathcal{T}[0, n+1, k]$
```

Theorem 9.1 *MinRowAlg solves* MinRow *in $O(K^\star \cdot n^3)$ time, where $K^\star$ is the number of rows in the optimal* MinRow *solution.*

Proof. The running time of MinRowAlg follows immediately. The outer loop is iterated $K^\star$ times, and in each iteration of the outer loop we perform $O(n^2)$ iterations of the nested inner loops. In each iteration it takes linear time to find the minimum of the set $\Theta^k_{i,j}$, which contains $O(n)$ elements.



(a) $X_j = x_j$



(b) $X_j = X_\ell + W_j$

Figure 9.4: Two ℓ-compact $(i, j, 3)$-labelings.

It remains to prove the correctness of MinRowAlg by showing $\mathcal{T}[i, j, k] = \mathcal{X}_{i,j,k}$ for all $0 \le i \le j \le n+1$ and $1 \le k \le K^\star$. The proof is by induction over k and for each k by induction over $j - i$.

For $k = 1$ we use SingleRowAlg described in Section 9.3.1 and it follows immediately that the entries $\mathcal{T}[\cdot, \cdot, 1]$ are correct.

Next we prove correctness for $k > 1$. In the base case $i = j$ the algorithm sets $\mathcal{T}[i, i, k] = x_i$, which is the x-value $\mathcal{X}_{i,i,k}$ of a compact (i, i, k)-labeling that simply consists of the single label l_i placed in its leftmost possible position $X_i = x_i$ in row $Y_i = k$.

Now let's assume $i < j$ and let $\mathcal{L}$ be an (i, j, k)-labeling. By definition l_i and l_j are in row k. This implies that there is a well-defined *predecessor* l_ℓ of l_j in row k for $i \le \ell < j$; we say that l_ℓ *precedes* l_j in row k. We call a label l_ℓ a *feasible* predecessor of l_j if there exists an (i, j, k)-labeling, where l_ℓ precedes l_j. Let $F(i, j, k)$ be the set of all feasible predecessors of l_j in an (i, j, k)-labeling. Now we define an (i, j, k)-labeling $\mathcal{L}$ to be *ℓ-compact* if l_ℓ precedes l_j and there is no other (i, j, k)-labeling with predecessor l_ℓ where the position of l_j is further to the left. Figure 9.4 shows two ℓ-compact $(i, j, 3)$-labelings.

predecessor
precedes
feasible
ℓ-compact

Every compact (i, j, k)-labeling $\mathcal{L}$ is also ℓ-compact for the predecessor l_ℓ of l_j in $\mathcal{L}$ since by definition of a compact (i, j, k)-labeling, there is no other (i, j, k)-labeling where the position of l_j is further to the left. On the other hand the ℓ-compact (i, j, k)-labeling with the leftmost position of l_j over all feasible predecessors $l_\ell \in F(i, j, k)$ is a compact (i, j, k)-labeling, since there is no other (i, j, k)-labeling where the position of l_j is further to the left. For every ℓ-compact (i, j, k)-labeling the leftmost x-position of l_j is $X_j^\ell = \max\{x_j, X_\ell + W_j\}$, since l_j must be W_j to the right of its predecessor but cannot be left of x_j, see Figure 9.4. Hence the x-position of a compact (i, j, k)-labeling $\mathcal{X}_{i,j,k} = \min\{X_j^\ell \mid l_\ell \in F(i, j, k)\}$. Note that if $F(i, j, k) = \emptyset$ we obtain $\mathcal{X}_{i,j,k} = \infty$. We claim that $\Theta_{i,j}^k = \{X_j^\ell \mid l_\ell \in F(i, j, k)\}$ and thus MinRowAlg correctly computes $\mathcal{T}[i, j, k] = \min \Theta_{i,j}^k = \mathcal{X}_{i,j,k}$.

Let $\mathcal{L}$ be an ℓ-compact (i, j, k)-labeling and let $\mathcal{L}_{\text{left}}$ be the (i, ℓ, k)-labeling formed by labels $l_i, \dots, l_\ell$ of $\mathcal{L}$. We can assume that $\mathcal{L}_{\text{left}}$ is compact since otherwise we can replace $\mathcal{L}_{\text{left}}$ in $\mathcal{L}$ by a compact (i, ℓ, k)-labeling that actually constrains the position of l_j *less* than $\mathcal{L}_{\text{left}}$. Hence $X_j^\ell = \max\{x_j, \mathcal{X}_{i,\ell,k} + W_j\}$. Since $\ell < j$ we obtain from the induction hypothesis that $\mathcal{X}_{i,\ell,k} = \mathcal{T}[i, \ell, k]$, that is, the values in $\Theta_{i,j}^k$ are actually $X_j^\ell = \max\{x_j, \mathcal{T}[i, \ell, k] + W_j\}$.

It remains to show that a value X_j^ℓ is in $\Theta_{i,j}^k$ if and only if l_ℓ is a feasible predecessors in $F(i, j, k)$. For the only-if-part, let l_ℓ be a feasible predecessor in $F(i, j, k)$. Then obviously $i \le \ell < j$. Moreover, $\mathcal{X}_{i,\ell,k} = \mathcal{T}[i, \ell, k] \le x_j$ since otherwise l_j would be pushed too far to the right for being part of an (i, j, k)-labeling. Finally, we observe that any ℓ-compact (i, j, k)-labeling induces a labeling $\mathcal{L}_{\text{right}}$ of the labels $l_{\ell+1}, \dots, l_{j-1}$, in which they are restricted to lie to the right of x_ℓ, to the left of x_j and below row k. We can extend $\mathcal{L}_{\text{right}}$ to an $(\ell, j, k-1)$-labeling by placing l_ℓ at $X_\ell = x_\ell$, $Y_\ell = k-1$ and l_j at $X_j = x_j + W_j$, $Y_j = k-1$; see Figure 9.5. This implies $\mathcal{T}[\ell, j, k-1] < \infty$.

For the if-part, let $i \le \ell < j$ such that $\mathcal{T}[i, \ell, k] \le x_j$ and $\mathcal{T}[\ell, j, k-1] < \infty$. We combine a compact (i, ℓ, k)-labeling $\mathcal{L}_{\text{left}}$ (exists because $\mathcal{T}[i, \ell, k] \le x_j < \infty$) and labels $l_{\ell+1}, \dots, l_{j-1}$

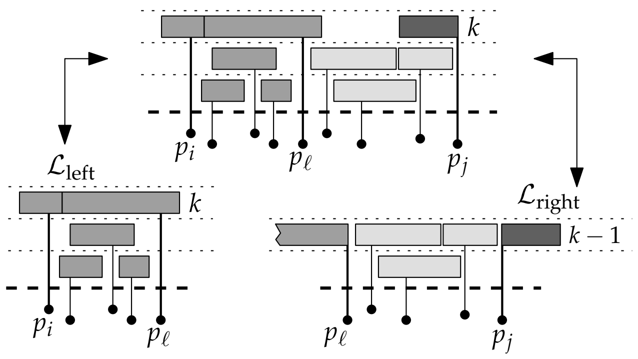


Figure 9.5: (De-)composition of an ℓ-compact (i, j, k)-labeling into an (i, ℓ, k)-labeling and an $(\ell, j, k-1)$-labeling.

from a compact $(\ell, j, k-1)$-labeling with the label l_j at position $X_j = x_j + W_j$ in row k as illustrated in Figure 9.5. This yields a feasible (i, j, k)-labeling $\mathcal{L}$ since $\mathcal{L}_{\text{left}}$ is feasible and labels $l_{\ell+1}, \ldots, l_{j-1}$ lie below row k, to the right of x_ℓ, and to the left of x_j. We know that $\mathcal{T}[i, \ell, k] < x_j$ and hence l_j can be placed at $x_j + W_j$ without overlapping l_ℓ. Furthermore, both l_i and l_j are in row k and l_ℓ precedes l_j, that is, l_ℓ is a feasible predecessor in $F(i, j, k)$. This concludes the proof of correctness. □

Note that $K^\star$ is in $O(n)$, hence the worst-case time complexity of MinRowAlg is $O(n^4)$.

9.3.3 *Weight Maximization*

In this section we first show NP-hardness of MaxWeight. Then, we present a pseudo-polynomial time algorithm showing that MaxWeight is actually only weakly NP-hard. Recall that for a MaxWeight instance we are given, in addition to the set P of anchor points and the set L of corresponding labels, for each point p_i in P a positive integer weight w_i, and we are also given a positive integer K. The problem then asks for a feasible labeling $\mathcal{L} = (S, \pi)$ that maximizes the total weight $\sum_{l_i \in S} w_i$ among all feasible labelings that use at most K rows.

Theorem 9.2 MaxWeight *is NP-hard, even for $K = 1$.*

Proof. Our proof is inspired by an NP-hardness proof for an internal 4-slider point labeling problem by Garrido et al. [95, Theorem 3]. It is by reduction from the following NP-hard variant of Partition [91]: Given a set $A = \{a_1, a_2, \ldots, a_{2m}\}$ and a size $s(a) \in \mathbb{Z}^+$ for each $a \in A$, is there a subset $A' \subseteq A$ such that $\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)$ and A' contains exactly one of a_{2i-1}, a_{2i} for every $1 \leq i \leq m$? For each Partition-instance I we construct a MaxWeight-instance J such that the weight of an optimal solution to J allows us to decide whether or not I is a yes-instance of Partition. We specify the set P of points, their weights, and the widths of their labels in J as illustrated in Figure 9.6. More precisely, we define a large constant $C = 1000 \sum_{a \in A} s(a)$. Next we define (from left to right) points $p_L, p_1, \ldots, p_{2m}, p_R$ on a horizontal line and their distances as $|\overline{p_L p_1}| = |\overline{p_{2m} p_R}| = C/2$, $|\overline{p_{2i-1} p_{2i}}| = (s(a_{2i-1}) + s(a_{2i}))/2$, and $|\overline{p_{2i} p_{2i+1}}| = C$. The corresponding labels have widths $W_L = W_R = |\overline{p_L p_R}|$ and $W_i = C + s(a_i)$. Furthermore, we define $w_L = w_R = W_L$ and $w_i = W_i$, that is, the weight of a point is equal to the width of its label.

A solution to the Partition instance I exists if and only if a feasible labeling in a single row with total weight $3\,\overline{p_L p_R}$ exists. To see why, we first assume that we are given such a labeling. Clearly, this labeling contains the label for p_L and the label for p_R. Moreover, the space between p_L and p_R is completely filled with labels for other points. Since the labels are extremely wide (compared to the distances $\overline{p_{2i-1} p_{2i}}$) exactly one of the labels l_1, l_2 has to be selected and its leader has to lie roughly in the center of the label. This again implies

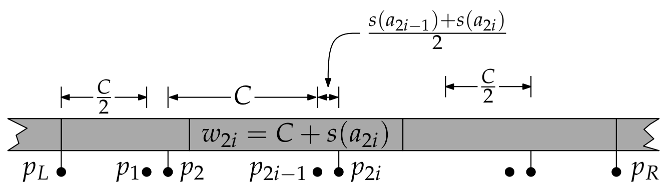


Figure 9.6: Reducing Partition to MaxWeight.

that exactly one of the labels l_3, l_4 has to be selected, and so on. By induction, it follows that for $i = 1, \dots, m$ exactly one of the labels l_{2i-1}, l_{2i} has to be selected. Since $\overline{p_L p_R}$ is exactly half the total width of labels $l_1, \dots, l_{2m}$ the labeled points correspond to a feasible partition. If, however, we are given a solution $A' \subseteq A$ to the Partition instance, we can construct the labeling by selecting labels p_L, p_R plus the labels for the points corresponding to the elements in A'. It is easy to see that by applying algorithm SingleRowAlg from Section 9.3.1 to the selected labels, we obtain a feasible labeling. □

Since MaxWeight is NP-hard, we cannot hope for a polynomial-time algorithm that solves it, unless P = NP. But our reduction uses labels with extremely large weights. Hence, we propose a pseudo-polynomial time algorithm, that is, an algorithm whose running time is polynomial in n, k, and the numeric value of $\sum_{i=1}^{n} w_i$, but exponential in the length of the encoding of $\sum_{i=1}^{n} w_i$. If the label weights are small integer numbers such an algorithm can still be fast in practice.

pseudo-polynomial time algorithm for maxweight. We first extend our notation from Section 9.3.2. Recall that for a labeling $\mathcal{L}$ the set S contains all labels selected for display. For a pair of indices $0 \le i \le j \le n+1$, a row index k, and a weight c we define an (i,j,k,c)-labeling as a feasible labeling of $P_{i,j}$ such that $l_i, l_j \in S$ and $Y_i = Y_j = k$, all other labels $l_\ell \in S$ are placed in row k or below (i.e., $Y_\ell \le k$), and the total weight of the labels in S is c. Analogously, we say an (i,j,k,c)-labeling is *compact* if there is no other (i,j,k,c)-labeling where l_j has a smaller x-position. Again, we call the x-position of l_j in a compact (i,j,k,c)-labeling $\mathcal{L}$ the x-position of $\mathcal{L}$. Note that this definition generalizes our definition of a compact (i,j,k)-labeling, since an (i,j,k,c)-labeling is an (i,j,k)-labeling if $c = \sum_{\ell=i}^{j} w_\ell$.

In our algorithm MaxWeightAlg for MaxWeight we add a fourth dimension to the table $\mathcal{T}$ that allows us to distinguish labelings of different weights. Let $w_{\text{total}} = \sum_{i=1}^{n} w_i$ be the total weight of all points in P. Then, each entry $\mathcal{T}[i,j,k,c]$ stores the x-position of a compact (i,j,k,c)-labeling, where $0 \le i \le j \le n+1$, $1 \le k \le \lceil n/2 \rceil$, and $0 \le c \le w_{\text{total}}$. The maximum-weight labeling using K rows can be constructed by backtracking from $\mathcal{T}[0,n+1,K,c_{\max}]$, where $c_{\max} = \max\{c \mid \mathcal{T}[0,n+1,K,c] < \infty\}$.

As before, we compute $\mathcal{T}$ in a bottom-up fashion with respect to the topmost row k, the weight c, and the distance $j - i$. We set $\mathcal{T}[i,i,k,w_i] = x_i$ for all k and $\mathcal{T}[i,i,k,c] = \infty$ for all k and $c \ne w_i$. In all other cases we use the recurrence $\mathcal{T}[i,j,k,c] = \min \Theta_{i,j}^{k,c}$, where $\Theta_{i,j}^{k,c}$ for $k = 1$ is defined as the set

$$\Theta_{i,j}^{1,c} = \left\{\max\left\{x_j, \mathcal{T}[i,\ell,1,a] + W_j\right\} \mid i \le \ell < j,\, \mathcal{T}[i,\ell,1,a] \le x_j,\, a = c - w_j\right\} \tag{9.2}$$

and for $k > 1$ as the set

$$\Theta_{i,j}^{k,c} = \left\{\max\left\{x_j, \mathcal{T}[i,\ell,k,a] + W_j\right\} \mid \begin{array}{c} i \le \ell < j,\, \mathcal{T}[i,\ell,k,a] \le x_j, \\ \mathcal{T}[\ell,j,k-1,b] < \infty,\, a = c - b + w_\ell \end{array}\right\}. \tag{9.3}$$

We give the pseudo code for MaxWeightAlg in Algorithm 4.

Theorem 9.3 *MaxWeightAlg solves* MaxWeight *in* $O(Kn^3 w_{\text{total}}^2)$ *time.*

Algorithmus 4 : MaxWeightAlg

```
initialize $\mathcal{T}$
for $j = 1$ to $n+1$ do
    for $i = 0$ to $j-1$ do
        for $c = 0$ to $w_{\text{total}}$ do
            $\mathcal{T}[i,j,1,c] = \min \Theta_{i,j}^{1,c}$
for $k = 2$ to $K$ do
    for $j = 1$ to $n+1$ do
        for $i = 0$ to $j-1$ do
            for $c = 0$ to $w_{\text{total}}$ do
                $\mathcal{T}[i,j,k,c] = \min \Theta_{i,j}^{k,c}$
reconstruct solution from $\mathcal{T}[0, n+1, K, c_{\max}]$
```

Proof. The algorithm MaxWeightAlg is similar to MinRowAlg, but uses four instead of three nested loops to compute the $O(Kn^2 w_{\text{total}})$ entries of $\mathcal{T}$. Each entry is computed as the minimum of a set $\Theta_{i,j}^{k,c}$ containing $O(nw_{\text{total}})$ elements. This yields an overall running time of $O(Kn^3 w_{\text{total}}^2)$.

We now show the correctness of the algorithm analogously to the proof of Theorem 9.1 but taking the weight constraints into account. For the case $i = j$ and arbitrary k it is easy to see that the x-value of a compact (i,i,k,c)-labeling is the leftmost possible position x_i of l_i if $c = w_i$ and ∞ otherwise.

Before we consider the general case, we extend the notion of ℓ-compact as introduced in Section 9.3.2. For given weights $a, c \in \mathbb{N}$ with $a \geq c$, we call a pair (l_ℓ, a) a feasible predecessor pair of (l_j, c) if there exists an (i,j,k,c)-labeling ($i < j$), where l_ℓ precedes l_j and the total weight of the selected labels $S \cap \{l_i, \ldots, l_\ell\}$ is a. We define $F(i,j,k,c)$ as the set of all feasible predecessors pairs of (l_j, c) in an (i,j,k,c)-labeling. We then define an (i,j,k,c)-labeling $\mathcal{L}$ to be (ℓ,a)-compact if (l_ℓ, a) precedes (l_j, c) and there is no other (i,j,k,c)-labeling $\mathcal{L}'$ where (l_ℓ, a) precedes (l_j, c) and which has smaller x-position than $\mathcal{L}$. As before it is clear that every compact (i,j,k,c)-labeling $\mathcal{L}$ is also (ℓ,a)-compact for the predecessor pair (l_ℓ, a) of (l_j, c). Conversely, the (ℓ,a)-compact (i,j,k,c)-labeling with smallest x-position over all feasible predecessor pairs $(l_\ell, a) \in F(i,j,k,c)$ is compact. For every (ℓ,a)-compact (i,j,k,c)-labeling $\mathcal{L}$ the x-position of $\mathcal{L}$ is $X_j^{\ell,a} = \max\{x_j, X_\ell + W_j\}$. Note that the value X_ℓ depends implicitly on a since the value of X_ℓ depends on $\mathcal{L}$ which directly depends on a. To ease notation we omit this parameter in the description of X_ℓ. The x-position of a compact (i,j,k,c)-labeling is $\min\{X_j^{\ell,a} \mid (l_\ell, a) \in F(i,j,k,c)\}$. Our claim is that $\Theta_{i,j}^{k,c} = \{X_j^{\ell,a} \mid (l_\ell, a) \in F(i,j,k,c)\}$ and thus the algorithm is correct.

Now, let3 $\mathcal{L}$ be an (ℓ,a)-compact (i,j,k,c)-labeling and let $\mathcal{L}_{\text{left}}$ be the induced (i,ℓ,k,a)-labeling. As in the proof of Theorem 9.1 we can assume that $\mathcal{L}_{\text{left}}$ is compact and hence by the induction hypothesis $X_j^{\ell,a} = \max\{x_j, \mathcal{T}[i,\ell,k,a] + W_j\}$, that is, the values in $\Theta_{i,j}^{k,c}$ are actually $X_j^{\ell,a}$ for some pairs (ℓ, a).

It remains to show that $X_j^{\ell,a} \in \Theta_{i,j}^{k,c}$ if and only if the pair $(l_\ell, a) \in F(i,j,k,c)$. We start with the case $k = 1$. If $(l_\ell, a) \in F(i,j,1,c)$, then obviously $i \leq \ell < j$ and also $\mathcal{T}[i,\ell,1,a] \leq x_j$ since otherwise l_j cannot be in a feasible position. Moreover, for the total weight of the labeling to be c, the weight of the labels in $S \cap \{l_i, \ldots, l_\ell\}$ must be $c - w_j$, since in a single row l_j is the only label to the right of l_ℓ. If, on the other hand, the three constraints for the set $\Theta_{i,j}^{1,c}$ hold, we can combine a compact $(i,\ell,1,a)$-labeling (which exists because $\mathcal{T}[i,\ell,1,a] < \infty$) with weight a and the label l_j with weight w_j placed in row 1 at position $X_j = x_j + W_j$ into an $(i, j, 1, c)$-labeling for $c = a + w_j$. Therefore, (l_ℓ, a) is indeed a feasible predecessor pair in $F(i, j, 1, c)$.

In the general case for $k > 1$ the argument is similar to $k = 1$. Let first $(l_\ell, a) \in F(i, j, k, c)$ be a feasible predecessor pair. Then $i \leq \ell < j$ and $\mathcal{T}[i, \ell, k, a] \leq x_j$ as before, but additionally any (ℓ, a)-compact labeling induces a labeling $\mathcal{L}_{\text{right}}$ of labels $S \cap \{l_{\ell+1}, \dots, l_{j-1}\}$ that is strictly below row k, to the right of x_ℓ and to the left of x_j. Furthermore, $\mathcal{L}_{\text{right}}$ has weight $c - a - w_j$. Again, we extend $\mathcal{L}_{\text{right}}$ to an $(\ell, j, k-1, b)$-labeling by placing l_ℓ and l_j in row $k-1$ with x-positions $X_\ell = x_\ell$ and $X_j = x_j + W_j$, similar to the situation depicted in Figure 9.5. Note that the weight b of this labeling is $b = c - a - w_j + w_\ell + w_j$ so that all constraints put on set $\Theta_{i,j}^{k,c}$ are satisfied.

Conversely, if all constraints for $\Theta_{i,j}^{k,c}$ are satisfied we can compose a feasible (i, j, k, c)-labeling $\mathcal{L}$ from a compact (i, ℓ, k, a)-labeling $\mathcal{L}_{\text{left}}$ and a compact $(\ell, j, k-1, b)$-labeling $\mathcal{L}_{\text{right}}$ as sketched in Figure 9.5. The labels in $S \cap \{l_i, \dots, l_\ell\}$ are placed as in $\mathcal{L}_{\text{left}}$, the labels in $S \cap \{l_{\ell+1}, \dots, l_{j-1}\}$ as in $\mathcal{L}_{\text{right}}$ and l_j at $x_j + W_j$. The weights of the sub-labelings are chosen such that the weight of $\mathcal{L}$ correctly adds up to c. □

9.3.4 *Label Number Maximization*

In this section we present an algorithm to solve MaxLabels, that is, to place as many labels as possible in K given rows. Note that Theorem 9.3 of the previous section directly yields a polynomial time algorithm if we set $w_i = 1$ for all $1 \leq i \leq n$. However, this would result in a running time of $O(Kn^5)$. Here we show that there is a faster algorithm which uses an exchange argument based on the fact that all labels have the same weight.

We first introduce an adapted notation for cardinality-maximal labelings. We define a *cm-(i, j, k)-labeling* to be a feasible labeling of a subset $\hat{P} \subseteq P_{i,j}$ in rows 1 to k with both l_i and l_j placed in row k such that there is no feasible labeling of another subset $\bar{P} \subseteq P_{i,j}$ with the same properties but $|\bar{P}| > |\hat{P}|$. We say a cm-(i, j, k)-labeling $\mathcal{L}$ is *cm-compact* if there is no other cm-(i, j, k)-labeling with smaller x-position.

cm-(i, j, k)-labeling

cm-compact

We will compute two three-dimensional tables $\mathcal{T}$ and $\mathcal{K}$. An entry $\mathcal{T}[i, j, k]$ for $0 \leq i \leq j \leq n+1$ and $1 \leq k \leq K$ stores the x-position of a cm-compact cm-(i, j, k)-labeling. We note that $\mathcal{T}$ can no longer contain the value ∞ since for any triple (i, j, k) there is always a feasible labeling with $\hat{P} = \{p_i, p_j\}$ and their labels placed disjointly in row k. An entry $\mathcal{K}[i, j, k]$ stores the actual cardinality of a cm-(i, j, k)-labeling.

The recursive definitions of $\mathcal{T}$ and $\mathcal{K}$ are as follows:

$$\mathcal{T}[i, j, k] = \begin{cases} x_i & \text{if } i = j \\ \min \Theta_{i,j}^k & \text{otherwise} \end{cases}, \tag{9.4}$$

with

$$\Theta_{i,j}^k = \left\{ \max \{x_j, \mathcal{T}[i, \ell, k] + W_j\} \;\middle|\; \begin{array}{c} i \leq \ell < j,\ \mathcal{T}[i, \ell, k] \leq x_j, \\ \mathcal{K}[i, j, k] = \mathcal{K}[i, \ell, k] + \mathcal{K}[\ell, j, k-1] - 1 \end{array} \right\}, \tag{9.5}$$

$$\mathcal{K}[i, j, k] = \begin{cases} 1 & \text{if } \; i = j \\ \max \kappa_{i,j}^k & \text{otherwise} \end{cases}, \tag{9.6}$$

with

$$\kappa_{i,j}^k = \{\mathcal{K}[i, \ell, k] + \mathcal{K}[\ell, j, k-1] - 1 \mid i \leq \ell < j,\ \mathcal{T}[i, \ell, k] \leq x_j\}. \tag{9.7}$$

The dynamic programming algorithm MaxLabelsAlg for MaxLabels computes $\mathcal{K}$ and $\mathcal{T}$ in a bottom-up fashion analogously to our previous algorithms. Note that for each triple

i, j, k we first compute $\mathcal{K}[i, j, k]$ and then based on that $\mathcal{T}[i, j, k]$. The final solution contains $\mathcal{K}[0, n+1, K] - 2$ labels and can be obtained by backtracking from $\mathcal{T}[0, n+1, K]$. We give the pseudo code for MaxLabelsAlg in Algorithm 5.

Algorithmus 5 : MaxLabelsAlg

```
initialize $\mathcal{T}$
for $i = 0$ to $n+1$ do
    $\mathcal{T}[i, i, k] = x_i$
    $\mathcal{K}[i, i, k] = 1$
for $k = 1$ to $K$ do
    for $j = 1$ to $n+1$ do
        for $i = 0$ to $j-1$ do
            $\mathcal{K}[i, j, k] = \max \kappa^k_{i,j}$
            $\mathcal{T}[i, j, k] = \min \Theta^k_{i,j}$
reconstruct solution from $\mathcal{T}[0, n+1, K]$
```

Theorem 9.4 MAXLABELS *can be solved by dynamic programming in $O(Kn^3)$ time.*

Proof. The running time of the dynamic programming algorithm follows from the fact that the tables are both of size $O(Kn^2)$ and computing each entry consists of finding the minimum or maximum of a set of $O(n)$ elements.

The correctness proof follows exactly the same arguments about the decomposition of a cm-(i, j, k)-labeling $\mathcal{L}$ into two labelings $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ by splitting $\mathcal{L}$ at the predecessor l_ℓ of l_j in row k (Figure 9.5). The definition of $\Theta^k_{i,j}$ implies that a value $X^\ell_j = \max\{x_j, \mathcal{T}[i, \ell, k] + W_j\}$ is contained in the set if and only if it leads to a cm-(i, j, k)-labeling; this is achieved by requiring $\mathcal{K}[i, j, k] = \mathcal{K}[i, \ell, k] + \mathcal{K}[\ell, j, k-1] - 1$. Note that here we need to subtract 1 because label l_ℓ is counted twice otherwise. The set $\kappa^k_{i,j}$ contains the cardinalities of all feasible ($\mathcal{T}[i, \ell, k] \leq x_j$) compositions of a cm-compact cm-(i, ℓ, k)-labeling and a cm-compact cm-$(\ell, j, k-1)$-labeling. Entry $\mathcal{K}[i, j, k]$ is then simply the maximum value in $\kappa^k_{i,j}$.

The interesting aspect for showing the correctness is the following exchange argument. Assume that there is a cm-(i, j, k)-labeling $\mathcal{L}$ with predecessor l_ℓ of l_j but $\mathcal{T}[i, \ell, k] > x_j$, that is, the x-position of this labeling is not contained in $\Theta^k_{i,j}$. Since $\mathcal{T}[i, \ell, k] > x_j$ it follows that the sub-labeling $\mathcal{L}'$ for $P_{i,\ell}$ induced by $\mathcal{L}$ is not cardinality-maximal (recall that $\mathcal{T}[i, \ell, k]$ stores the smallest x-position among all cm-(i, j, k)-labelings). So in order to have l_ℓ as the predecessor of l_j some other label left of l_ℓ in row k must be removed from the selected labels, that is, we lose at least one label from S. But since all labels are worth the same, we can just as well remove l_ℓ itself from S and use label $l_{\ell'}$, the predecessor of l_ℓ in a cm-compact cm-(i, ℓ, k)-labeling, as l_j's predecessor. Since $l_{\ell'}$ is the predecessor of l_ℓ in that labeling we know $\mathcal{T}[i, \ell', k] \leq x_\ell < x_j$, so that the x-position $X^{\ell'}_j$ is in fact contained in $\Theta^k_{i,j}$.

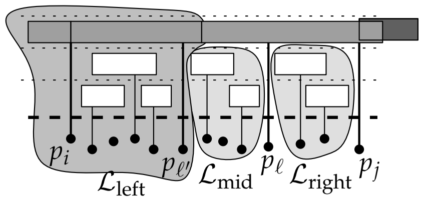

p_i
$p_{\ell'}$
$\mathcal{L}_{\text{left}}$
$\mathcal{L}_{\text{mid}}$
p_ℓ
$\mathcal{L}_{\text{right}}$
p_j

Figure 9.7: Construction of a labeling for Theorem 9.4.

It remains to argue that this labeling with predecessor $l_{\ell'}$ is at least as good as $\mathcal{L}$, that is, $\mathcal{K}[i,j,k] \geq |\mathcal{L}|$. By the arguments presented above, we know that there is an (i,j,k)-labeling where $l_{\ell'}$ is the predecessor of l_j. Combining this with the definition of $\kappa^k_{i,j}$ we obtain:

$$\mathcal{K}[i,j,k] \geq \mathcal{K}[i,\ell',k] + \mathcal{K}[\ell',j,k-1] - 1 \tag{9.8}$$

Next, we consider the (i,ℓ,k)-labeling with $l_{\ell'}$ as predecessor of l_ℓ and ignore the label l_ℓ from it. We split this labeling into two parts. We denote the labeling for the points from p_i to $p_{\ell'}$ by $\mathcal{L}_{\text{left}}$, and the labeling for the points from $p_{\ell'+1}$ to $p_{\ell-1}$ by $\mathcal{L}_{\text{mid}}$; see Figure 9.7. Finally, there is a labeling $\mathcal{L}_{\text{right}}$ for the points from $p_{\ell+1}$ to p_{j-1}. It is easy to see that $|\mathcal{L}_{\text{right}}|$ is at most $\mathcal{K}[\ell,j,k-1] - 2$. Now, consider both $\mathcal{L}_{\text{mid}}$ and $\mathcal{L}_{\text{right}}$. By removing the leader of p_ℓ separating $\mathcal{L}_{\text{mid}}$ from $\mathcal{L}_{\text{right}}$ we obtain

$$\mathcal{K}[\ell',j,k-1] \geq \mathcal{K}[\ell',\ell,k-1] + \mathcal{K}[\ell,j,k-1] - 2 \tag{9.9}$$

Combining equations (9.8) and (9.9) yields

$$\mathcal{K}[i,j,k] \geq \mathcal{K}[i,\ell',k] + \mathcal{K}[\ell',\ell,k-1] + \mathcal{K}[\ell,j,k-1] - 3 \tag{9.10}$$

Recall that ℓ' is the label in the (i,ℓ,k)-labeling that is the predecessor of l_ℓ in row k. Hence, by definition of $\mathcal{K}[i,\ell,k]$ it follows that $\mathcal{K}[i,\ell',k] + \mathcal{K}[\ell',\ell,k-1] - 1 \leq \mathcal{K}[i,\ell,k]$. Plugging this into equation (9.10) yields:

$$\mathcal{K}[i,j,k] \geq \mathcal{K}[i,\ell,k] - 1 + \mathcal{K}[\ell,j,k-1] - 1 \tag{9.11}$$

As we have already argued, the labeling that is induced by $\mathcal{L}$ for the points p_i to p_ℓ cannot be cardinality maximal. Hence, the labeling for those points can contribute at most $\mathcal{K}[i,\ell,k] - 1$ labels to $\mathcal{L}$. The labeling induced by $\mathcal{L}$ for the remaining points can contribute at most $\mathcal{K}[\ell,j,k-1] - 1$ labels. Finally, we can conclude

$$\mathcal{K}[i,j,k] \geq |\mathcal{L}| \tag{9.12}$$

This means that our algorithm indeed finds a labeling that has at least as many labels as $\mathcal{L}$. □

9.3.5 *Two-sided Panorama Labeling*

In this section we consider the two-sided panorama labeling problem. In this variation of panorama labeling we allow rows above and below the panorama we want to label; for an example see Figure 9.8. Unfortunately, by a result of Garrido et al. [95, Theorem 3] it follows directly, that MinRow, MaxLabels, and MaxWeight are NP-hard in the two-sided panorama labeling model, even if the problems are restricted to one row per side.

Corollary 9.2 *For two-sided panorama labeling the problems* MinRow, MaxLabels, *and* MaxWeight *are* NP*-hard.*

For still finding a two-sided panorama labeling we propose in the following a mixed-integer linear (MILP) program formulation that can handle MinRow, MaxLabels, and MaxWeight.

MIXED-INTEGER LINEAR PROGRAMMING. In the following we give a MILP formulation that solves a given instance of two-sided panorama labeling. We begin by describing a MILP formulation for MaxLabels, and proceed by sketching how to extend this formulation such that it can handle MaxWeight and MinRow. For simplicity we assume, without loss of generality, that the x-coordinates of the input points are all non-negative. We can transform an instance with negative x-coordinates to an instance with strictly positive x-coordinates by simply shifting all points sufficiently far to the right.

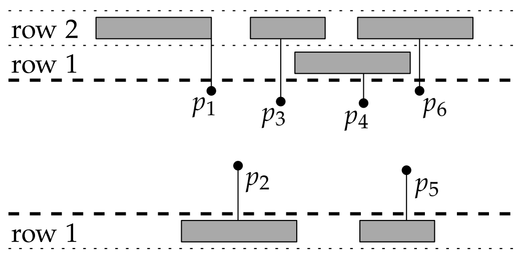


Figure 9.8: Example for a two-sided panorama labeling

Recall that for MAXLABELS we need to find the maximum number of labels which can be placed into K rows. For the two-sided case we denote the number of rows that are above the horizon by K_a and the number of rows below the panorama by K_b.

For each label l_i and each row $1 \le r \le K_a$ above the horizon we introduce a binary variable a_i^r, and for each row $1 \le r' \le K_b$ we introduce a binary variable $b_i^{r'}$. The desired meaning of the variables is that, if a variable a_i^r ($b_i^{r'}$) has value 1, then the label l_i is placed in row r (r'), and a_i^r ($b_i^{r'}$) has value 0 if the label l_i is not placed in this row. Further, we introduce for every label l_i a continuous variable X_i that stores the x-position of l_i, if l_i is selected.

We are now ready to introduce the constraints of the MILP. First we introduce a constraint that ensures that a label is placed in at most one row. Then, we give two constraints which in combination guarantee that each label is connected to its leader; see requirement **(F1)** in Section 9.2. For each label l_i we require:

$$\sum_{1\le r\le K_a} a_i^r + \sum_{1\le r'\le K_b} b_i^{r'} \le 1 \tag{9.13}$$

$$X_i \ge x_i \tag{9.14}$$

$$X_i \le x_i + W_i \tag{9.15}$$

To generate a valid panorama labeling, we need to ensure that no two labels intersect (see requirement **(F3)**), that is, we require $X_i \le X_j - W_j$ for every pair of labels l_i and l_j placed in the same row with $i < j$. However, this constraint must not influence the position of pairs of labels in different rows. To ensure this we introduce a large constant M which we use to "activate" or "deactivate" the constraint. This is a common trick used in formulating MILPs [54]. In the following, we set M to be the largest x-coordinate of the input points plus the largest width of all labels among the input. Since there is in principle no difference between placing labels above the horizon or below it we explicitly introduce the constraints only for placing labels *above* the horizon in the following (constraints 9.16–9.18). The constraints for placing labels *below* the horizon are obtained by replacing each occurrence of a_i^r, a_j^ℓ, and K_a with b_i^r,b_j^ℓ, and K_b, respectively.

We introduce for each pair of labels l_i, l_j with $i < j$ and for every r, $1 \le r \le K_a$, the following constraint:

$$X_i \le X_j - W_j + (1 - a_i^r)M + (1 - a_j^r)M \tag{9.16}$$

Note that this constraint has only an effect on X_i and X_j if both a_i^r and a_j^r are set to 1, that is, both l_i and l_j are placed in the same row (then the constraint is the desired $X_i \le X_j - W_j$). Otherwise, the value on the right-hand side of the equation is always larger than the value of the left-hand side.

We also need to guarantee that no label can intersect a leader; see requirement **(F2)**. First, we introduce a constraint that ensures that no label can intersect the leader of a label to its right. For a label l_i placed in row r we need the constraint $X_i \le x_j$ for all labels l_j,

$i < j$ whose labels are placed in rows above r. Making use of the constant M as above we introduce for every pair of labels l_i, l_j with $i < j$ and for every r, $1 \le r \le K_\mathrm{a}$, the following constraint:

$$X_i \le x_j + (1 - \mathrm{a}_i^r)M + \left(1 - \sum_{r < \ell \le K_\mathrm{a}} \mathrm{a}_j^\ell\right) M \tag{9.17}$$

Similarly, to ensure that the label l_i does not intersect the leader of a label l_j to its left ($j < i$) that is placed in a row above l_i we need the constraint $X_i - W_i \ge x_j$. We introduce for every pair of Labels l_i, l_j with $j < i$ and for every r, $1 \le r \le K_\mathrm{a}$, the following constraint:

$$X_i - W_i \ge x_j - (1 - \mathrm{a}_i^r)M - \left(1 - \sum_{K_\mathrm{a} \ge \ell > r} \mathrm{a}_j^\ell\right) M \tag{9.18}$$

The constraints 9.17 and 9.18 influence X_i only if both $\mathrm{a}_i^r = 1$ (that is, the label l_i is placed in row r) and the sum of all a_j^ℓ, $\ell > r$ is 1 (that is, the label l_j is placed in a row above l_i).

Subject to all introduced constraints we maximize the objective function:

$$\sum_{1 \le i \le n} \left(\sum_{1 \le r \le K_\mathrm{a}} \mathrm{a}_i^r + \sum_{1 \le r' \le K_\mathrm{b}} \mathrm{b}_i^{r'} \right)$$

Extending the above formulation such that it can handle MaxWeight can be done straightforwardly by modifying the objective function slightly to $\sum_{1 \le i \le n}((\sum_{1 \le r \le K_\mathrm{a}} \mathrm{a}_i^r + \sum_{1 \le r' \le K_\mathrm{b}} \mathrm{b}_i^{r'}) \cdot w_i)$.

We can extend the above formulation such that it can solve MinRow by adding additional constraints and modifying the objective function. Recall that for MinRow the number of rows necessary to find a solution is bounded by $\lceil n/2 \rceil$. Hence, we can set $K_\mathrm{a} = K_\mathrm{b} = \lceil n/2 \rceil$. Since for this problem all labels need to be placed we need to modify constraint 9.13 to the following.

$$\sum_{1 \le r \le K_\mathrm{a}} \mathrm{a}_i^r + \sum_{1 \le r' \le K_\mathrm{b}} \mathrm{b}_i^{r'} = 1 \tag{9.19}$$

Further, we add a binary variable z_r for each row above the horizon and a binary variable z'_r for each row below it. The desired meaning of the newly added variables is that if a row r contains at least one label, then, the corresponding variable z_r is 1. For each label l_i we require

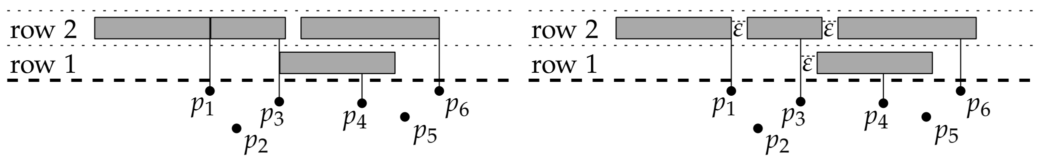
$$z_r \ge \mathrm{a}_i^r \qquad 1 \le r \le K_\mathrm{a} \tag{9.20}$$

$$z'_r \ge \mathrm{b}_i^{r'} \qquad 1 \le r' \le K_\mathrm{b} \tag{9.21}$$

Finally, the objective function we want to minimize is $\sum_{r \le K_\mathrm{a}} z_r + \sum_{r \le K_\mathrm{b}} z'_r$. Since we minimize the objective function, we ensure that every z_r is set to 0 if it is possible, that is, if the row r does not contain a label. Note that although this formulation minimizes the number of rows in which the labels are placed, it might contain empty rows which can be removed in a post-processing step.

9.4 EXTENSIONS

In this section we sketch several extensions to our algorithms. With some extensions we aim at handling additional aesthetic constraints, for example, to ensure that two labels or a label and a leader of another label do not come too close. With this particular constraint the problem MinRow can become infeasible if the distance between two input points is too small. Similar problems can arise with other extensions. We do not address these problems in detail; often they can be revealed and possibly resolved in a pre-processing step.



(a) Result of MaxLabelsAlg. (b) Result of MaxLabelsAlg with label spacing.

Figure 9.9: Result of MaxLabelsAlg of a MaxLabels instance for two rows with and without label spacing.

9.4.1 *Label Spacing*

Our dynamic programming algorithms ensure that no two labels intersect and that no label intersects a leader other than its own. It is possible, however, that two labels (or a label and a leader) get arbitrarily close to each other. At first glance there seems to be an obvious solution to the problem: simply enlarge the labels by an arbitrary buffer amount. However, this can result in labels that are disconnected from their leaders. To see this consider a label that is in its leftmost/rightmost position. Since we increased the width of the label its x-position may be to the left of its own leader, which results in a label that is disconnected from its leader. Hence, we propose a different approach. To avoid this problem, we add the requirement that the horizontal distance between two labels in the same row (or between a label and the leader of a label in a higher row) must not be smaller than a user-defined value $\varepsilon \geq 0$; see Figure 9.9. MinRowAlg ensures this requirement if we add $+\varepsilon$ to the definition of equation (9.1) at the appropriate positions. We need it to ensure that no label l_ℓ is considered for which the distance between the right border of l_i and the anchor of l_ℓ has distance less than ε. This is ensured by requiring $\mathcal{T}[i,\ell,k] + \varepsilon \leq x_j$. Further, we need that if a label l_ℓ is placed right of l_i in the k-th row, it has distance of at least ε. We obtain this by requiring $\max\{x_j, \mathcal{T}[i,\ell,k] + W_j + \varepsilon\}$. This modification of equation (9.1) yields

$$\Theta_{i,j}^{k} = \left\{\max\{x_j, \mathcal{T}[i,\ell,k] + W_j + \varepsilon\} \mid i \leq \ell < j,\ \mathcal{T}[i,\ell,k] + \varepsilon \leq x_j,\ \mathcal{T}[\ell,j,k-1] < \infty\right\}.$$

For MaxWeightAlg similar modifications are necessary. We modify equations (9.2) and (9.3). The first equation is for the recurrence relation for the first row (that is, $k = 1$)

$$\Theta_{i,j}^{1,c} = \left\{\max\left\{x_j, \mathcal{T}[i,\ell,1,a] + W_j + \varepsilon\right\} \mid i \leq \ell < j,\ \mathcal{T}[i,\ell,1,a] + \varepsilon \leq x_j,\ a = c - w_j\right\}$$

and for $k > 1$

$$\Theta_{i,j}^{k,c} = \left\{\max\left\{x_j, \mathcal{T}[i,\ell,k,a] + W_j + \varepsilon\right\} \mid \begin{array}{c} i \leq \ell < j,\ \mathcal{T}[i,\ell,k,a] + \varepsilon \leq x_j, \\ \mathcal{T}[\ell,j,k-1,b] < \infty,\ a = c - b + w_\ell \end{array}\right\}.$$

Finally, the modifications required for MaxLabelsAlg are in principle identical to those desribed before. We need to modify the equations (9.5) and (9.7) yielding:

$$\Theta_{i,j}^{k} = \left\{\max\left\{x_j, \mathcal{T}[i,\ell,k] + W_j + \varepsilon\right\} \;\middle|\; \begin{array}{c} i \leq \ell < j,\ \mathcal{T}[i,\ell,k] + \varepsilon \leq x_j, \\ \mathcal{K}[i,j,k] = \mathcal{K}[i,\ell,k] + \mathcal{K}[\ell,j,k-1] - 1 \end{array}\right\},$$

$$\kappa_{i,j}^{k} = \left\{\mathcal{K}[i,\ell,k] + \mathcal{K}[\ell,j,k-1] - 1 \mid i \leq \ell < j,\ \mathcal{T}[i,\ell,k] + \varepsilon \leq x_j\right\}.$$

9.4.2 *Horizontal Space Constraints*

In Section 9.3 we gave algorithms to compute labelings that use a small number of rows but we did not care about space consumption in the horizontal direction. If we want to ensure,

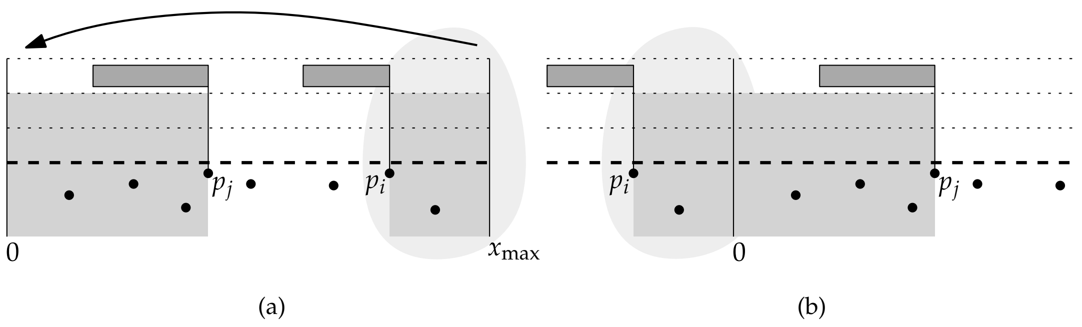


Figure 9.10: Example of 360°-panoramas before (a) and after (b) transformation.

for example, that no label intersects the left or right border of the panorama, we have to restrict the sliding of labels. This can be done by changing the definition of the two dummy points p_0 and p_{n+1} in Section 9.3.2, that is, we set $x_0 = x_{\min}$ and $x_{n+1} = x_{\max}$, where $x_{\min}$ ($x_{\max}$) is the smallest (largest) allowed x-coordinate for the left (right) boundary of a label. Then, a $(0, n+1, k)$-labeling with the minimum feasible value for k is an optimal solution to the horizontally constrained version of MinRow.

Note that in this case Lemma 9.1 is not true anymore. There are MinRow instances for which more than $\lceil n/2 \rceil$ rows are necessary, and there are even instances for which no solution is possible. However, it is easy to see that no MinRow instance that has a solution requires more than n rows. Hence, the worst-case time complexity of MinRowAlg is still in $O(K^\star n^3)$, where $K^\star$ is in $O(n)$. If an instance of MinRow has no valid solution, then $\mathcal{T}[0, n+1, n] = \infty$. The algorithms MaxLabelsAlg and MaxWeightAlg can be adapted by the same technique.

9.4.3 *360°-panoramas*

We consider a 360°-panorama as an image curved around the inside of a cylinder, thus it does not have a left or right boundary. We can extend our algorithm for MinRow to handle such images. Our extension is based on the following observation: if there is a solution to MinRow with k rows, there is also a solution with k rows that contains one of the labels l_i at position $X_i = x_i$, $Y_i = k$. A trivial approach is thus to solve the problem for $i = 1, 2, \dots, n$, each time splitting the cylindric image at x_i to obtain a conventional image. Since we have to solve the problem n times the asymptotic running time increases by one order of magnitude. With a simple change, however, *one* execution of our MinRow-algorithm suffices.

Our idea is to extend table $\mathcal{T}$, which so far contained values $\mathcal{T}[i, j, k]$ only for $i \leq j$. The new version of the algorithm computes values $\mathcal{T}[i, j, k]$ *for all* $1 \leq i, j \leq n$ and $1 \leq k \leq K^\star$, where $K^\star$ is the smallest feasible number of rows. If $j < i$, a labeling corresponding to $\mathcal{T}[i, j, k]$ simply contains labels $l_{i+1}, \dots, l_n$ and $l_1, \dots, l_j$. To ensure that the solutions for $l_{i+1}, \dots, l_n$ and $l_1, \dots, l_j$ are compatible we need to transform the x-coordinates of points p_ℓ with $\ell > i$ to $x_\ell - x_{\max}$, where $x_{\max}$ is the width of the panorama. This transformation essentially moves the points p_ℓ to the left of $x = 0$; see Figure 9.10. We also need to transform the appropriate values in $\mathcal{T}$ by the same method.

We compute $\mathcal{T}[i, j, k]$ as before, ignoring the fact that in a 360°-panorama l_i and l_j may intersect in the interval bounded to the left by x_j and to the right by x_i. If our algorithm determines $\mathcal{T}[i, j, k] < \infty$ we need to ensure that there exists a feasible solution where l_i and l_j do not intersect. This, however, can be done by a simple sweep over all labels that are placed in row k in the solution. With this change, MinRowAlg still needs $O(K^\star n^3)$ time. The algorithms MaxLabelsAlg and MaxWeightAlg can be adapted the same way.

9.4.4 *Elite Labels*

For the most important sites, for example, the Willis Tower in the panorama of Chicago (see Figures 9.1 and 9.12), we may want to guarantee that the solution contains the corresponding labels. We term such a label an *elite label*. Now we can ask for a feasible labeling that contains all elite labels plus as many other labels as possible in a given number k of rows. Assuming that such a solution exists, we can apply our algorithm for MAXWEIGHT. We simply have to set the weight of each elite label to a large enough number (for example, $n+1$) and the weight of each non-elite label to one. With this approach we have $w_{\text{total}} \in O(n^2)$, thus the algorithm requires $O(kn^7)$ time. With the following simple modifications of recurrences (9.6) and (9.4), however, we can reuse our $O(kn^3)$-time algorithm for MAXLABELS.

Each $(i, j, 1)$-labeling with $j > i$ contains a label l_ℓ preceding l_j. We have to avoid that we omit an elite label between l_i and l_j. To do so, we exchange the condition $i \leq \ell < j$ in the definitions of $\kappa^1_{i,j}$ and $\Theta^1_{i,j}$ with $e \leq \ell < j$, where l_e is the last elite label among $l_{i+1}, \dots, l_{j-1}$. Obviously, we do not lose any feasible solution with this. Moreover, the modification suffices since our algorithm always constructs a solution based on feasible one-row solutions for smaller (sub-)instances. Similarly we can extend our algorithm for MAXWEIGHT to find a labeling in k rows containing all elite labels plus other labels of maximum total weight.

9.4.5 *Optimizing Label Positions*

A secondary criterion for aesthetically pleasing panorama labelings is that the labels are as close as possible to being centered above their respective sites. So far we did not consider this criterion in our problem definitions. In fact, the dynamic programming algorithms described in Section 9.3 place all labels at their leftmost feasible positions. We can remedy this unwanted side effect by subsequently fine-tuning the horizontal label positions. We traverse each row k of a given feasible labeling from right to left. Let S_k be the set of labels in row k. The general idea is that during the traversal, we shift a label to the right if it decreases the overall cost $H_k = \sum_{l_i \in S_k} |\Delta_i|$, where $\Delta_i = x_i - (X_i - W_i/2)$, which means that $|\Delta_i|$ is the horizontal distance between the center $X_i - W_i/2$ of label l_i and the anchor point p_i. Note that a right shift of a label l might require a right shift of the label directly to the right of l.

Since this approach does not change the row assignment of the labels we only need to ensure that while shifting the labels no label intersects another label or leader. This suffices to guarantee that the labeling remains valid. In the following we describe a simple $O(n^2)$-time algorithm that minimizes H_k by centering the labels as much as possible above their respective anchor points. The general idea is to identify maximal sets of touching labels in the same row called *chains* and to repeatedly move subsets of the chains to the right.

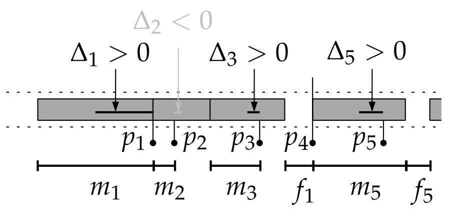
chains

In the first step of the algorithm we assign each label to a chain. Two labels that are in the same row are assigned to the same chain if their borders touch each other. If a label touches no other label it defines a singleton chain. We say that the *suffix* of a label l_i consists of l_i and all labels in its chain that are to the right of l_i.

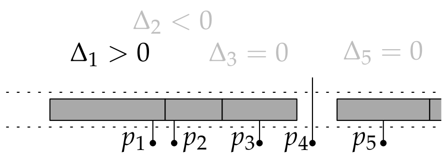
suffix

The general idea behind our algorithm is to iteratively move one suffix at a time to the right in order to improve the value of the objective function H_k. We distinguish between four types of *events* which may occur while moving a suffix: (i) a label in the suffix becomes centered above its anchor, (ii) a label in the suffix is moved to its rightmost position, (iii) the right border of the rightmost label in the suffix touches the leader of another label in a row above, or (iv) the rightmost label in the suffix touches a label that belongs to another chain. The first type of event occurs when a positive Δ_i becomes 0, and the second type occurs when the horizontal distance m_i between a label's left border and its anchor becomes 0, that

events


$\Delta_2 < 0$
$\Delta_1 > 0$
$\Delta_3 > 0$
$\Delta_5 > 0$
p_1 p_2 p_3 p_4 p_5
m_1 m_2 m_3 f_1 m_5 f_5

(a) Before applying our post-processing algorithm.



(b) After applying the post-processing algorithm.

Figure 9.11: Illustrations for our post-processing algorithm. In both illustrations two chains are shown.

is, $m_i = x_i - (X_i - W_i) = 0$. Since the row assignment of the solution is fixed we can easily determine in a pre-processing step for each label whether it has to its right a label, a leader or nothing. This allows us to determine in a later step the distance f_i that the suffix of l_i can be moved until an event of the third or fourth type occurs. For an illustration of these concepts see Figure 9.11.

Now, in a single row, the algorithm traverses the chains from right to left beginning with the rightmost chain. In each chain it performs a second-level traversal from right to left. In this second-level traversal our algorithm determines for each label l_i the values for Δ_i, m_i, and the difference d_i between the number of positive and non-positive offsets Δ_j in its suffix. Note that if d_i of a suffix is positive, it means moving the suffix to the right decreases H. The algorithm also computes for each label l_i how far its suffix can be moved to the right until the first event occurs by determining $M_i = \min\{\Delta_j, m_j, f_i \mid \Delta_j > 0, l_j \text{ in suffix of } l_i\}$.

Finally, we need to select the suffix of the currently considered chain that we actually move to the right. After determining the value of d_i for each label we choose the label with maximum positive d_i. If there is more than one label with maximum d_i we choose the left-most label among them. If no label has a positive d_i we move on to the next chain since a shift to the right would increase H_k. Now, consider we have selected a label l_i with maximum positive d_i in its chain. We move the whole suffix of l_i by M_i to the right. Note that if l_i is not the leftmost label in its chain, this splits the chain into two separate chains. Further, if after moving the suffix of l_i the rightmost label of the suffix touches the left-most label of another chain (that is, an event of the fourth type has occurred) we have merged two independent chains into a new one. If, after moving l_i's suffix, the suffix cannot be moved further to the right (that is, an event of type (ii) or (iii) has occurred) we move on to the next chain directly left of the current chain. Otherwise, we repeat the second-level traversal for the current chain.

Note that the output of our dynamic programming algorithms described in Section 9.3 ensures that no label can initially be moved to the left. Now, to see that the algorithm minimizes $H = \sum_{l_i \in S} |\Delta_i|$ consider a single row. It is clear that after a chain has been considered by the algorithm it does no longer contain a suffix that would decrease H if moved further to the right. Since this is true for all chains of a each row the correctness of the algorithm follows. It remains to argue that the running time is $O(n^2)$. Consider again a single row, say the k-th row (which contains $|S_k|$ labels). Determining the values of Δ_i, d_i, m_i and M_i for each label $l_i \in S_k$ can be done in constant time per label. Every time we move a label at most $O(|S_k|)$ labels in its suffix are moved with it. Since for each label at most three events can occur there are at most $3|S_k|$ move operations, each of which requires $O(|S_k|)$ time. Hence, for the k-th row our algorithm requires $O(|S_k|^2)$ time. The total running time of our algorithm is thus $O(\sum_k |S_k|^2)$ which is in $O(n^2)$.

9.4.6 *Minimizing the Total Leader Length*

Since long leaders produce visual clutter we may want place all labels while minimizing the total length of their leaders. Our algorithm for MaxWeight can be modified to solve this problem. We simply have to define the weight w_i for a label l_i depending on the row in which we place l_i. We define $w_i = n + 1 - Y_i$, which implies that a larger weight is assigned to positions closer to the horizon. Our setting ensures $w_i > 0$ since n is an upper bound on the number of rows required for an optimal solution. Using our algorithm for MaxWeight we can solve the problem in $O(n^5 w_{\text{total}}^2) = O(n^9)$ time.

9.4.7 *Fixed Order k-position Feature Labeling*

Sometimes we do not want to label one distinct point, but we rather want to label a feature that is represented by an area (for example, a building) where each point inside this area is equally representative of the feature we want to label. The flexibility of choosing a representative anchor point might help increase the quality of the labeling (for example, fewer rows are required to display all labels). Since in our setting the problem is defined only by the x-coordinates of the anchor points, each area feature reduces to a horizontal line segment or interval spanning the horizontal width of the area. Note that these intervals may overlap. However, in most labeling instances the objects to be labeled have a natural fixed order (for example, the order of buildings is implicitly given by their centroids), and a labeling should reflect this order. Hence, we may assume that there is a given total order on the anchor points.

Unfortunately, it is difficult to adapt our algorithms to handle anchors that can freely move along a horizontal line segment. We can, however, solve a slightly less general variant of this problem. If instead of line segments, we restrict the possible positions of each anchor to z distinct points we can solve MinRow, and MaxLabels in polynomial time with respect to n and z.

More specifically, for each label l_i we are given a set of z points $p_i^1 = (x_i^1, -1)$, $p_i^2 = (x_i^2, -1), \ldots, p_i^z = (x_i^z, -1)$ with x-coordinates $x_i^1, x_i^2, \ldots, x_i^z$ (for simplicity we assume that all y-coordinates are set to -1). Further, to reflect the order of the features we aim to label, we are also given a total order on the anchors of the labels. More specifically, the anchor of a Label l_i is always placed to the left of l_j's anchor for all $j > i$. Now, we are ready to describe how to adapt MinRowAlg. First, we extend the definition of a compact (i, j, k)-labeling to a compact (i, g, j, h, k)-labeling, where $1 \leq g, h \leq z$, p_i^g and p_j^h are the two anchors for l_i, and l_j, respectively. The definition of $\mathcal{X}_{i,g,j,h,k}$ can be extended the same way. We increase the size of the table $\mathcal{T}$ by two dimensions. An entry $\mathcal{T}[i, g, j, h, k]$ stores the x-value $\mathcal{X}_{i,g,j,h,k}$ of a compact (i, g, j, h, k)-labeling, and some additional backtracking information. Each entry $\mathcal{T}[i, g, j, h, k]$ is obtained by computing $\min \Theta_{i,g,j,h}^k$, where $\Theta_{i,g,j,h}^k$ for $1 \leq i < j \leq n$, $1 \leq g, h \leq z$, and $k \geq 1$ is defined as the set

$$\Theta_{i,g,j,h}^k = \left\{ \max\{x_j^h, \mathcal{T}[i, g, \ell, f, k] + W_j\} \;\middle|\; \begin{array}{c} i \leq \ell < j, \\ \mathcal{T}[i, g, \ell, f, k] \leq x_j^h, \\ \mathcal{T}[\ell, f, j, h, k-1] < \infty, \\ 1 \leq f, g, h \leq z, \\ x_i^g \leq x_l^f < x_j^h \end{array} \right\}.$$

The correctness follows by the same argumentation as in the proof of Theorem 9.1. The running time of this algorithm is $O(K^\star n^3 z^3)$. For small values of z the factor z^3 can still be considered a constant.

The extension of MaxLabelsAlg, and MaxWeightAlg to handle multiple possible positions can be done by following the same basic idea.

9.5 EXPERIMENTAL RESULTS

In this section we evaluate our algorithms, which we implemented in C++ using GTK+ and Cairo for the visual output. We tested the algorithms both for real-world instances as well as randomly generated instances. For a discussion of a case study with the Chicago skyline see Figure 9.12.

We executed the experiments on a single core of an Intel Xeon E5-2670 processor that is clocked at 2.66 GHz. The machine is running Linux 3.4.28-2.20 and has 64 GiB of RAM. We compiled our C++ implementation with GCC 4.7.1, using optimization level 3.

For each set of test parameters we generated 1,000 test instances with a width of 1280 pixels. All input points have integer x-coordinates between 0 and 1279. These coordinates were chosen uniformly at random. The label widths were randomly chosen from a Gaussian distribution with mean 115 and standard deviation $\sigma = 20$. These values stem from real-world experience with geographic place names.

MINROW. The first set of our experiments focuses on our implementation of MinRowAlg. Table 9.1 reports the minimum, average, and maximum running time for labeling instances with varying numbers of labels. We observe that even for instances where the points lie unreasonably dense (for example, instances with 150 labels where the average distance between points is roughly 8 pixels) our algorithm solves all instances in less than 0.25s. Interestingly, the vast majority of instances with 75 or more labels is formed by worst-case instances for MinRow, that is, they require $\lceil n/2 \rceil$ rows. Thus for reasonable real-world instances we can expect that our algorithm generates an optimal solution near-instantaneously.

Table 9.1: Running time results of our implementation of MinRow for instances with 10–150 labels.

Input	running time [ms]		
#labels	min.	avg.	max.
10	< 1	< 1	< 1
25	< 1	< 1	< 1
50	1	4	5
75	6	12	17
100	21	33	44
125	68	91	97
150	165	198	204

MAXLABELS. In the second set of experiments we investigate the running times of our implementation of MaxLabelsAlg. We believe that in most practical application no more than four rows of labels are used and hence ran the first part of experiments for $K = 4$. We observe from the data in Table 9.2 that our algorithm performs again very well, even for large instances. In this case it even outperforms the previous MinRow algorithm. This is mainly due to the fact that many instances are worst-case instances for the MinRow algorithm.

Our algorithm is also able to quickly generate solutions for less realistic instances where we allow the labels to be placed in at most 50 rows, see Table 9.2. We observe that even in the worst-case scenario, that is, placing 150 labels in at most 50 rows, the execution time is less than 260ms.

MAXWEIGHT. Next, we evaluate our MaxWeight algorithm. For this we use two different weight distributions. In some use cases it might be useful to define a ranking of the label importance. Then, for a set of n labels, we assign each label a distinct integer weight between 1 and n. In all generated instances the weights were distributed uniformly at random. A different way of determining label weights is to group labels into classes of equal importance. Generally, we can expect that there is only a limited number of such

Table 9.2: Running time results of our implementation of MaxLabels with $K = 4$ and $K = 50$ rows for instances with 10–150 labels.

Input	running time [ms]					
	$K = 4$			$K = 50$		
#labels	min.	avg.	max.	min.	avg.	max.
10	< 1	< 1	< 1	< 1	< 1	< 1
25	< 1	< 1	< 1	< 1	1	3
50	< 1	< 1	2	7	8	13
75	2	3	7	24	25	29
100	5	6	12	63	66	83
125	10	11	20	125	133	150
150	17	18	28	214	238	260

Table 9.3: Running time results of our implementation of MaxWeight for instances with 10–75 ranked labels and with 10–100 labels in four importance classes with weights $\{1, 2, 4, 8\}$.

Input	running time [s]					
	ranked labels			grouped labels		
#labels	min.	avg.	max.	min.	avg.	max.
10	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
25	0.23	0.26	0.28	0.05	0.10	0.14
30	0.74	0.82	0.91	0.14	0.23	0.31
40	4.51	5.27	5.89	0.53	0.84	1.18
50	21.49	23.25	26.00	1.47	1.92	2.46
75	306.47	327.42	363.60	12.55	15.00	17.63
100	-	-	-	51.34	58.12	62.70

classes. In the experiment we defined four classes and the labels were assigned to the classes uniformly at random. Labels in class i have weight 2^i.

We report the running times of our implementation of MaxWeightAlg for both types of weight distributions in Table 9.3. These results were again generated using $K = 4$ rows. Note that the measured execution times in these tables are reported in seconds and not milliseconds as before.

Since the algorithm has a pseudo-polynomial running time the higher execution times compared to our other algorithms were expected. Although the results reported in both tables confirm this expectation, we observe that for small and medium numbers of rows and labels, the algorithm still runs within an acceptable time frame. However, if we raise the number of available rows substantially or increase the total number of labels, the execution time grows quickly and may become unacceptable in practice.

TWO-SIDED PANORAMA LABELING. Finally, we have also implemented the MILP formulation for MaxLabels for the two sided case in C++ using the MILP solver Gurobi 5.10.

We set the number of rows above the panorama $K_a = 3$ and the number of rows below the panorama to $K_b = 1$ since in most visual depictions of such panorama labelings, the majority of labels is above the horizon and only a minority is placed below the picture.

For small instances (at most 25) labels, the optimum solution can be obtained quickly, such that it is sufficient for real-time applications, most of the time. However, increasing the number of labels to 30 already requires several seconds on average. Increasing the number of labels to 40 yields running times well above the 10 minute mark, which we set as timeout and which are, unfortunately, impractical; see Table 9.4 for detailed results.

Table 9.4: Running time results of our MILP-implementation for the two-sided MaxLabels problems for instances with 10–40 labels and $K_a = 3$ and $K_b = 1$.

Input	running time [s]		
#labels	min.	avg.	max.
10	< 0.01	0.01	0.01
25	0.02	0.16	1.10
30	0.02	8.86	38.74
40	> 600	> 600	> 600

EXTENSIONS. Here, we give a brief evaluation of some of the extension to our algorithms which we described in Section 9.4.

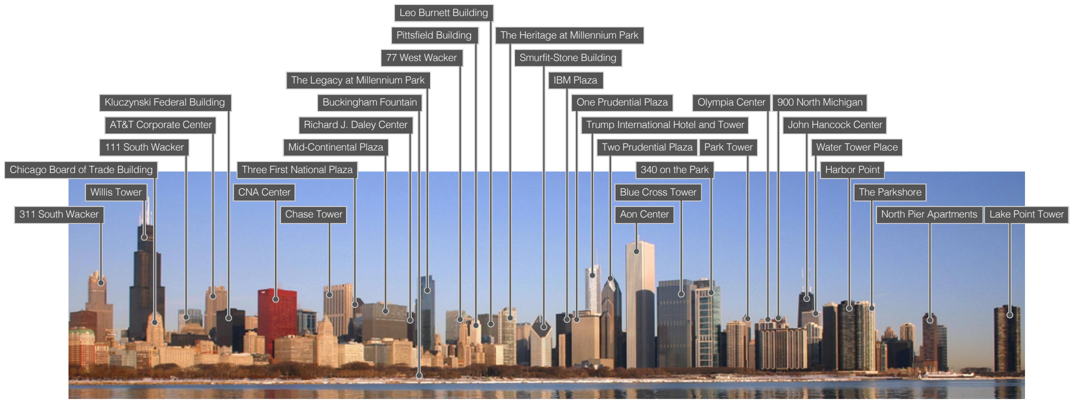
Unsurprisingly, the label spacing extension has no significant impact on the running time of the algorithms. For sensible values (for example, $\varepsilon = 10$) the impact on the number of displayed labels, or rows necessary to display all labels is negligibly.

Since we aim to produce visually appealing panorama labelings, we also implemented the algorithm that tries to optimize the position of each label in a post-processing step described in Section 9.4.5. The implementation is straightforward and for the tested instances the running time is even on the largest instances consisting of 150 labels less than 1ms. Since this step requires only little time we recommend to automatically apply this algorithm after computing a solution by one of our algorithms.

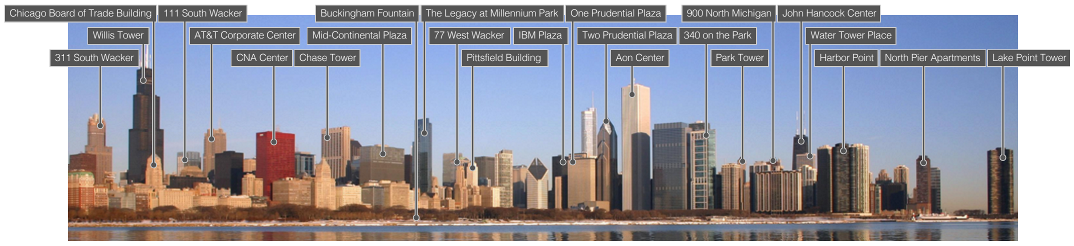
The last extension we implemented is the fixed order k-position feature labeling described in Section 9.4.7. We have implemented this extensions for both MinRowAlg as well as MaxLabelsAlg and we call the respective algorithms kPosMinRowAlg and kPosMaxLabelsAlg. For MinRowAlg and kPosMinRowAlg we compare the number of rows necessary to display all labels, and the time required for the algorithms to obtain the solution. For kPosMaxLabelsAlg we consider the number of labels the algorithm was able to place into $K = 4$ rows compared to the standard MaxLabelsAlg. We used the same generated instances as before, but added for each label four additional possible anchor points evenly spaced at 5 pixels. Two of the new possible anchor point positions were to the left and two wejo to the right of the initial anchor point position. Please note that we do not claim that the instances we tested resemble real-world data, but are only used to give a rough intuition on the algorithm's performance. We report the result of both implementations in Table 9.5. For convenience we also repeat the results of the original MinRowAlg and the MaxLabelsAlg.

Unfortunately, the results of the of the k-position MinRowAlg are not promising. The algorithm requires significantly more time than the original MinRowAlg algorithm and the number of rows necessary to display all labels decrease on average only slightly.

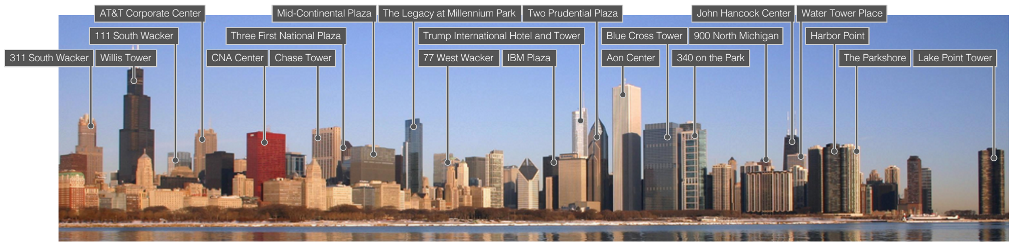
The results of kPosMaxLabelsAlg are slightly more promising. We observe that the number of labels the algorithm is able to place in 'the 4 rows is between 3% and 8% higher than in the standard MaxLabelsAlg algorithm (except for the small instances consisting of only 10 labels, where all labels can be always placed). Unfortunately, this comes with a significant increase in running time, but the algorithm might still be suitable for real-time applications for realistic instances. Usually, we expect less than 50 labels in a realistic instance which yields running times between 10 and 60ms.



(a) Solution of a MinRow instance that requires ten rows. We observe that the solution is aesthetically not appealing. In the center of the panorama the arrangement of labels is reminiscent of the worst-case scenario illustrated in Figure 9.3.



(b) Solution of a MaxLabels instance in three rows. Due to restricting the number of label rows, the result is much more pleasing than the MinRow solution (a). Of all 33 labels 23 are displayed. This indicates that only a few labels of densely placed points are responsible for the visually unpleasing MinRow result.



(c) Solution of a MaxWeight instance in three rows. We divided the buildings into four equal-size classes based on their height, that is, the $\lceil n/4 \rceil$ tallest buildings have weight 8, and the other classes have weights 4, 2, and 1 accordingly. This yields the maximum total weight of all labels $w_{\text{total}} = 128$. Of the 33 labels 23 labels are displayed and they have a total weight of 114. Although the two labelings in (b) and (c) look similar at first sight, note that for instance the MaxWeight solution contains a label for "Trump International Tower", the third tallest building in the panorama, while the MaxLabels solution misses that label. The MaxLabels solution achieves a total weight of 98.

Figure 9.12: A case study with the Chicago skyline using three of our algorithms. The input data for all three figures is the same. It consists a total of 33 labels. On a laptop clocked at 2.4 Ghz it took roughly 1ms to compute the panorama labelings in Figures (a), (b) and about 160ms to compute the labeling shown in (c). We used $\varepsilon = 10$px as label spacing, and applied the optimize label position extension. Photography: ©J. Crocker (`http://en.wikipedia.org/w/index.php?title=File:2010-02-19_16500x2000_chicago_skyline_panorama.jpg`).

Table 9.5: Evaluation of our implementation of kPosMinRowAlg and kPosMaxLabelsAlg algorithms with instances consisting of 10–150 labels. The average time in ms and the average number of rows/labels is reported.

Input	kPosMinRowAlg		MinRowAlg		kPosMaxLabelsAlg		MaxLabelsAlg	
#labels	∼time	∼rows	∼time	∼rows	∼time	∼labels	∼time	∼labels
10	< 1	2.21	< 1	2.28	1	10	< 1	10
25	2	6.01	< 1	6.04	10	24.10	< 1	23.36
50	36	19.71	4	19.76	60	36.73	2	34.09
100	662	50.18	33	50.29	472	45.02	13	41.38
150	3769	75.61	198	75.70	1641	48.78	28	44.84

9.6 CONCLUSION

We have presented polynomial time and practically fast algorithms for label placement using a new boundary-labeling model that allows multiple rows of sliding unit-height rectangular labels. In this model, each label is connected with its associated point by a vertical line segment. We have presented an $O(K^\star n^3)$-time algorithm for the basic problem MinRow, which minimizes the number of rows needed to place all labels. If the labeling is restricted to use at most K rows, however, we cannot generally place all labels. Therefore, we have investigated the problem MaxLabels, which aims at maximizing the total number of labels in K rows, and the problem MaxWeight, which aims at maximizing the total weight of labels in K rows. While MaxLabels can be solved in $O(Kn^3)$ time, MaxWeight turned out to be weakly NP-hard, yet can be solved by a pseudo-polynomial $O(Kn^3 w_{\text{total}}^2)$-time algorithm, where w_{total} is the total weight of all input labels. For the case of two-sided panorama labeling, we have proposed simple mixed-integer linear programming formulations.

According to our experiments, the algorithms for MinRow and MaxLabels are very fast for instances typically arising in practice, that is, they solve instances with up to 150 labels in less than one second. The algorithm for MaxWeight is fast if the weights are not too large. For example, with integer weights between 1 and 8, we can solve instances with 50 labels in less than two seconds. We think that our setting is realistic, since labeled images already with more than 50 labels quickly appear visually cluttered and more than 150 labels seems unrealistic in most cases. Similarly, if sites are assigned importance levels, there are usually few of them (for example, main landmarks, distinctive buildings, public buildings, other). We conclude that our dynamic programming algorithms can quickly produce visually pleasing labelings of real-world panorama images. For the two-sided panorama labeling our MILP formulation is fast only for relatively small instances. An interesting open question is how to handle labels of different heights that would take up more than one row or to investigate how to deal with area sites instead of point sites further than we did.

Although we considered several optimization problems in the context of panorama labeling, there are still many unanswered questions. For some applications it is of interest to investigate panorama labeling with labels that take up more than one row, which may be relevant for features with long names. As argued in Section 9.4.7, it is interesting to consider labelings of area features instead of point features, which do not have a unique horizontal order. In this paper we focused on point feature labeling and more research on other feature types is necessary. We have also discussed the two-sided panorama labeling problem and showed NP-hardness of the problem, but our proof (which is a simple reduction of a proof by Garrido et al. [95]) requires that the labels have non-uniform width. The computational complexity of two-sided panorama labeling problem with uniform width-labels is left open for future research. Finally, the vertical position of labels may have an influence on which features are observed first. In many cases higher features (mountain peaks, tall buildings) are more important and thus should have their labels in the top rows, while smaller features should use the lower rows. Minimizing leader length (Section 9.4.6) implicitly incorporates this idea, but it is interesting to find faster algorithms that optimize vertical positions explicitly, for example, taking weights of features into account as priorities to place their labels in higher rows.

Part III

CONCLUSION

CONCLUSION

This is the last chapter of this thesis, and we will briefly recapitulate the problems we considered and describe some open problems and next steps.

In the first part of thesis we discussed abstract maps and in particular schematizations. First, we investigated how to construct route sketches; see Chapter 4. We showed NP-hardness, gave an ILP formulation for optimal solutions and presented a dynamic programming approach for a restricted type of input paths. The experimental evaluation indicated that the approaches are fast enough for real-world applications and the resulting route sketches appear to be suitable for practical use. An interesting next step is to examine in, e. g., a user study, how these route sketches are perceived by users. Feedback there would be valuable and would possibly lead to a (hopefully small) modification of the model (this approach is similar to a concept called algorithm engineering [165]) where instead of performance guarantees (usually running time) we would give quality guarantees or æsthetic guarantees. Another extension that is interesting to examine would be the problem of creating destination maps which are another form of abstract maps and related to route sketches. Consider the following scenario: you are the owner of a restaurant and you want to display a simple picture on your website that shows how to get to your restaurant. This problem differs from the generation of route sketches in that only the destination is known and not the start. However, similarly to route sketches, the displayed part of the road network should be simplified and (probably) schematized. Although, there has been work done on this subject already, it has not been finally solved.

In the context of generating abstract maps we also considered the problem of generating the layouts for argument maps; see Chapter 5. We extended the topology-shape-metrics framework to generate column-based graph layouts. Although this topic is closely related to the field of graph drawing, which has enormous potential for new and useful algorithms for drawing graphs (and by that displaying information) we omit this since it is beyond the scope of this section. Instead we focus on the given problem of layouting argument maps. The approach we presented can only be seen as a first step for a sensible argument map layouter. Since argument reconstruction is a difficult process, a fully automatic layout solution seems to be very difficult to produce and maybe not even desireable. It is conceivable that a person analyzing arguments would want to have algorithms that are semi-automatic in that they allow the user to, say, fix some vertices to a position, or to give the algorithms hints on which arguments should be close to each other. Maybe a user would not want to have the same drawing style for the whole layout, e. g., some parts may be drawing with orthogonal edges, some parts may be better drawn using a radial layout, while still others may be best represented by a completely different structure. Besides this a more context aware drawing is also necessary. As stated in the introduction of this chapter there are mainly two types of arrows: support and attack. The layout algorithm in its current form does not use this information in any way. Finally, user interaction (adding/removing vertices or edges) should be supported by the layout algorithm such that the mental map of the user is maintained.

The other main topic of this thesis was labeling of maps or diagrams. We considered two dynamic scenarios, where dynamic means that continuous interactions with the map are possible, namely zooming and rotating, but viewed them separately. First, we discussed labeling of rotating maps (see Chapter 7), and evaluated the proposed methods. Then, we considered the sliding label model for zooming maps, but restricted ourselves to one-dimensional maps (e. g., timelines); see Chapter 8. These two approaches may serve only as a first step towards a more complete algorithmic approach for labeling dynamic maps. Real dynamic maps allow usually to rotate, zoom, pan and possibly tilt the map. These operations can usually be done completely independent from each other and how to incorporate labeling algorithms into a framework that supports all of these operations in a sound algorithmic manner has not yet been done, and, in my opinion is still far from being finished. Even how to properly model this problems is not entirely clear. Until then, either heuristic or meta heuristics need to be applied. Besides the problem of labeling dynamic maps, we briefly want to consider labeling of static maps here. Although we did not discuss this problem in this thesis we feel that since labeling dynamic maps is a more difficult case than labeling static maps, we also need to point out some future work in this area. At first glance much has been done in terms of algorithmic solutions for labeling static maps. However, most approaches focus only on point-feature labeling, while there is little work for line-feature or area labeling. This is unsatisfactory, since a properly labeled map usually consists of all three of those types. Even more problematic is that a combination of all three types of labeling (which is necessary for a properly labeled map) is also not in sight, and to my mind, there is also a significant amount of work to be done before this can be achieved. A first step, in my mind, is to properly label streets and combine this with point-feature labeling. However, labeling should not be done without the user of a map in mind. As with the route sketches, conducting a user study is of great importance to assess the quality of the output of the algorithms, and more importantly, the modeling. Finally, what is usually omitted when researching this problem from the theoretical computer science point of view is the actual context (besides point-features), e. g., ocean, and rivers. A context-aware labeling model (and algorithms based on this model) should also be one of the next steps in static map labeling.

Finally, we considered the problem of labeling panoramas and discussed several optimization problems in Chapter 9. For those we either proved NP-hardness and give ILP formulations to solve those problems optimally, or gave polynomial-time algorithms. This problem was done withing the framework of boundary labeling, which has been applied in the post in several different variations. However, we feel that a proper justification for some of the proposed models is still missing and more research into this is necessary. As basis for this we propose to research real-world applications of boundary labeling. One such example is the illustration of medical diagrams. There, mainly straight line leaders are used and not the (in the algorithmic community) popular *po*- or *opo*-leaders. A user study comparing these models seems to be an interesting subject.

All in all this thesis combines the work of several years of research into the topic of drawing and labeling maps. As can be seen by the above description the general topic is far from being fully researched and there are a lot of open problems that are yet to be solved. We hope that this thesis and some of the ideas presented in here can serve as basis for an inspiration of algorithmic approaches for computing better drawings of maps or diagrams, or produce better labeled maps.

BIBLIOGRAPHY

[1] Pankaj K. Agarwal, Marc van Kreveld, and Subhash Suri. "Label Placement by Maximum Independent Set in Rectangles." In: *Proc. 9th Canadian Conference on Computional Geometry (CCCG '97)*. 1997, pp. 233–238 (cited on pp. 85, 86).

[2] Pankaj K. Agarwal, Marc van Kreveld, and Subhash Suri. "Label Placement by Maximum Independent Set in Rectangles." In: *Computational Geometry: Theory and Applications* 11 (1998), pp. 209–218. DOI: 10.1016/S0925-7721(98)00028-5 (cited on pp. 85, 86, 95, 107, 132).

[3] Maneesh Agrawala and Chris Stolte. "Rendering Effective Route Maps: Improving Usability Through Generalization." In: *Proc. 28th Ann. Conf. on Computer Graphics and Interacttive Techniques (SIGGRAPH '01)*. ACM, 2001, pp. 241–249. DOI: 10.1145/383259.383286 (cited on p. 18).

[4] Kamran Ali, Knut Hartmann, and Thomas Strothotte. "Label Layout for Interactive 3D Illustrations." In: *Journal of WSCG* 13.1 (2005), pp. 1–8. URL: http://wscg.zcu.cz/wscg2005/Papers_2005/Journal/L07-full.pdf (cited on pp. 88, 131).

[5] Adriana C.F. Alvim and Éric D. Taillard. "POPMUSIC for the Point Feature Label Placement Problem." In: *European Journal of Operational Research* 192.2 (2009), pp. 396–413. DOI: 10.1016/j.ejor.2007.10.002 (cited on p. 87).

[6] Masatoshi Arikawa and Yahiko Kambayashi. "Dynamic Maps as Views of Geographic Databases." In: *Proc. 1st International Workshop on Mobile Multimedia Communications (MoMuc'93)*. 1993, pp. 1–4 (cited on p. 89).

[7] Esther M. Arkin and Ellen B. Silverberg. "Scheduling Jobs with Fixed Start and End Times." In: *Discrete Applied Mathematics* 18.1 (1987), pp. 1–8. DOI: 10.1016/0166-218X(87)90037-0 (cited on p. 146).

[8] Silvania Avelar and Matthias Müller. "Generating Topologically Correct Schematic Maps." In: *In Proc. 9th International Symp. Spatial Data Handling*. 2000 (cited on p. 1).

[9] Ken Been, Eli Daiches, and Chee Yap. "Dynamic Map Labeling." In: *IEEE Transactions on Visualization and Computer Graphics* 12.5 (2006), pp. 773–780. DOI: 10.1109/TVCG.2006.136 (cited on pp. 4, 88, 90, 93, 94, 95, 96, 131, 132, 133, 134).

[10] Ken Been, Martin Nöllenburg, Sheung-Hung Poon, and Alexander Wolff. "Optimizing Active Ranges for Consistent Dynamic Map Labeling." In: *Computational Geometry: Theory and Applications* 43.3 (2010), pp. 312–328. DOI: 10.1016/j.comgeo.2009.03.006 (cited on pp. 5, 88, 91, 93, 95, 96, 131, 132, 133, 134, 136, 138, 140).

[11] Michael A. Bekos, Michael Kaufmann, M. Nöllenburg, and Antonios Symvonis. "Boundary Labeling with Octilinear Leaders." In: *Algorithmica* 57.3 (2010), pp. 436–461. DOI: 10.1007/s00453-009-9283-6 (cited on p. 143).

[12] Michael A. Bekos, Michael Kaufmann, Katerina Potika, and Antonios Symvonis. "Multi-Stack Boundary Labeling Problems." In: *Proc. 26th Foundations of Software Technology and Theoretical Computer Science (FSTTCS '06)*. Vol. 4337. Lecture Notes in Computer Science. Springer Verlag, 2006, pp. 81–92. DOI: 10.1007/11944836_10 (cited on p. 143).

[13] Michael A. Bekos, Michael Kaufmann, Katerina Potika, and Antonios Symvonis. "Area-Feature Boundary Labeling." In: *Comp. J.* 53.6 (2010), pp. 827–841. DOI: 10.1093/comjnl/bxp087 (cited on p. 143).

[14] Michael A Bekos, Michael Kaufmann, and Antonios Symvonis. "Efficient Labeling of Collinear Sites." In: *Journal of Graph Algorithms and Applications* 12.3 (2008), pp. 357–380 (cited on p. 146).

[15] Michael A. Bekos, Michael Kaufmann, Antonios Symvonis, and Alexander Wolff. "Boundary Labeling: Models and Efficient Algorithms for Rectangular Maps." In: *Computational Geometry: Theory and Applications* 36.3 (2007), pp. 215–236. DOI: 10.1016/j.comgeo.2006.05.003 (cited on pp. 89, 90, 142, 143).

[16] Michael Bekos, Sabine Cornelsen, Martin Fink, Seok-Hee Hong, Michael Kaufmann, Martin Nöllenburg, Ignaz Rutter, and Antonios Symvonis. "Many-to-One Boundary Labeling with Backbones." In: *Proc. 13th International Symp. on Graph Drawing (GD '13)*. Vol. 8242. Lecture Notes in Computer Science. to appear. Springer Verlag, 2014 (cited on p. 143).

[17] Peter A. Beling. "Exact Algorithms for Linear Programming over Algebraic Extensions." In: *Algorithmica* 31.4 (2001), pp. 459–478. DOI: 10.1007/s00453-001-0049-z (cited on pp. 12, 32, 38).

[18] Blaine Bell, Steven Feiner, and Tobias Höllerer. "View Management for Virtual and Augmented Reality." In: *Proc 14th Ann. ACM Symp. on User Interface Software and Technology (UIST '01)*. ACM, 2001, pp. 101–110. DOI: 10.1145/502348.502363 (cited on pp. 88, 131).

[19] Marc Benkert, Herman Haverkort, Moritz Kroll, and Martin Nöllenburg. "Algorithms for Multi-Criteria Boundary Labeling." In: *Journal of Graph Algorithms and Applications* 13.3 (2009), pp. 289–317. DOI: 10.1007/978-3-540-77537-9_25 (cited on p. 143).

[20] Mark de Berg and Dirk H. P. Gerrits. "Approximation Algorithms for Free-Label Maximization." In: *Computational Geometry: Theory and Applications* 45.4 (2012), pp. 153–168. DOI: 10.1016/j.comgeo.2011.10.004 (cited on p. 87).

[21] Mark de Berg and Dirk H.P. Gerrits. "Labeling Moving Points with a Trade-Off between Label Speed and Label Overlap." In: *Proc. 21th Ann. European Symposium Algorithms (ESA '13)*. Vol. 8125. Lecture Notes in Computer Science. Springer Verlag, 2013, pp. 373–384. DOI: 10.1007/978-3-642-40450-4_32 (cited on pp. 87, 89).

[22] Mark Berg and Amirali Khosravi. "Optimal Binary Space Partitions in the Plane." In: *Proc. 6th Ann. International Conference on Computing and Combinatorics (COCOON 2010)*. Vol. 6196. Lecture Notes in Computer Science. Springer Verlag, 2010, pp. 216–225. DOI: 10.1007/978-3-642-14031-0_25 (cited on pp. 10, 20).

[23] Piotr Berman, Bhaskar DasGupta, S. Muthukrishnan, and Suneeta Ramaswami. "Efficient Approximation Algorithm for Tiling and Packing Problems with Rectangles." In: *Journal of Algorithms* 41.2 (2001), pp. 443–470. DOI: 10.1006/jagm.2001.1188 (cited on p. 85).

[24] Paola Bertolazzi, Giuseppe Di Battista, Carlo Mannino, and Roberto Tamassia. "Optimal Upward Planarity Testing of Single-Source Digraphs." In: *SIAM Journal of Computing* 27 (1998), pp. 132–169. DOI: 10.1137/S0097539794279626 (cited on p. 59).

[25] Gregor Betz. *Theorie Dialektischer Strukturen*. Klostermann, 2010 (cited on p. 74).

[26] Gregor Betz, Christof Doll, Andreas Gemsa, Ignaz Rutter, and Dorothea Wagner. "Column-based Graph Layouts." In: *Proc. 20th International Symp. on Graph Drawing*

(GD '12). Vol. 7704. Lecture Notes in Computer Science. Springer Verlag, 2013, pp. 236–247. DOI: 10.1007/978-3-642-36763-2_21 (cited on pp. 4, 55).

[27] Gregor Betz, Andreas Gemsa, Christof Mathies, Ignaz Rutter, and Dorothea Wagner. "Column-based Graph Layouts." In: *Journal of Graph Algorithms and Applications* 18.5 (2014), pp. 677–708. DOI: 10.7155/jgaa.00341 (cited on pp. 4, 55).

[28] Therese Biedl and Goos Kant. "A Better Heuristic for Orthogonal Graph Drawings." In: *Computational Geometry: Theory and Applications* 9.3 (1998), pp. 159–180. DOI: 10.1016/S0925-7721(97)00026-6 (cited on pp. 55, 61, 63, 76).

[29] Lenore Blum, Mike Shub, and Steve Smale. "On a Theory of Computation over the Real Numbers; NP Completeness, Recursive Functions and Universal Machines." In: *Proc. 29th Ann. Symp. Foundations of Computer Science (SFCS '88)*. SFCS '88. IEEE Computer Society, 1988, pp. 387–397. DOI: 10.1109/SFCS.1988.21955 (cited on p. 11).

[30] Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan. "Time Bounds for Selection." In: *Journal of Computer and System Sciences* 7 (4 1973), pp. 448–461. DOI: 10.1016/S0022-0000(73)80033-9 (cited on p. 63).

[31] Béla Bollobás. *Modern Graph Theory*. Graduate Texts in Mathematics. Springer Verlag, 1998 (cited on p. 7).

[32] Grady Booch, James Rumbaugh, and Ivar Jacobson. *Unified Modeling Language User Guide, The (2nd Edition)*. Addison-Wesley Professional, 2005 (cited on p. 58).

[33] Ravi Boppana and Magnús M. Halldórsson. "Approximating Maximum Independent Sets by Excluding Subgraphs." In: *BIT Numerical Mathematics* 32.2 (1992), pp. 180–196. DOI: 10.1007/BF01994876 (cited on p. 10).

[34] Ulrik Brandes, Markus Eiglsperger, Michael Kaufmann, and Dorothea Wagner. "Sketch-Driven Orthogonal Graph Drawing." In: *Proc. 10th International Symp. on Graph Drawing (GD '02)*. Vol. 2528. Lecture Notes in Computer Science. Springer Verlag, 2002, pp. 1–11. DOI: 10.1007/3-540-36151-0_1 (cited on p. 17).

[35] Ulrik Brandes and Boris Köpf. "Fast and Simple Horizontal Coordinate Assignment." In: *Proc. 9th International Symp. on Graph Drawing (GD '01)*. Vol. 2265. Lecture Notes in Computer Science. Springer Verlag, 2002, pp. 31–44. DOI: 10.1007/3-540-45848-4_3 (cited on p. 57).

[36] Ulrik Brandes and Barbara Pampel. "On the Hardness of Orthogonal-Order Preserving Graph Drawing." In: *Proc. 16th International Symp. on Graph Drawing (GD '08)*. Vol. 5417. Lecture Notes in Computer Science. Springer Verlag, 2009, pp. 266–277. DOI: 10.1007/978-3-642-00219-9_25 (cited on pp. 18, 19, 20, 32, 53).

[37] Ulrik Brandes and Barbara Pampel. "Orthogonal-Ordering Constraints are Tough." In: *Journal of Graph Algorithms and Applications* 17.1 (2013), pp. 1–10. DOI: 10.7155/jgaa.00281 (cited on pp. 18, 19, 20, 32, 53).

[38] Jürgen Branke. "Dynamic Graph Drawing." In: *Drawing Graphs: Methods and Models*. Vol. 2025. Lecture Notes in Computer Science. Springer Verlag, 2001. Chap. 9, pp. 228–246. DOI: 10.1007/3-540-44969-8_9 (cited on p. 17).

[39] Vasco Brattka and Peter Hertling. "Feasible Real Random Access Machines." In: *Journal of Complexity* 14.4 (1998), pp. 490–526. DOI: 10.1006/jcom.1998.0488 (cited on p. 11).

[40] Igor Brejc. *Maperitive*. 2011. URL: http://maperitive.net/ (cited on p. 82).

[41] Claus Brenner, Volker Paelke, Jan-Henrik Haunert, and Nora Ripperda. "The Geo-Scope – a Mixed-reality System for Planning and Public Participation." In: *Proc. 25th Urban Data Management Symposium (UDMS '06)*. 2006 (cited on p. 141).

[42] Llyod Arnold Brown. *The Story of Maps*. Dover Publications, 1979 (cited on p. 1).

[43] David J. Bryant, Margaret Lanca, and Barbara Tversky. "Spatial Concepts and Perception of Physical and Diagrammed Scenes." In: *Perceptual and Motor Skills* 81.2 (1995), pp. 531–546 (cited on p. 2).

[44] Kevin Buchin and Dirk H.P. Gerrits. "Dynamic Point Labeling is Strongly PSPACE-complete." In: *Proc. 24th Ann. International Symp. on Algorithms and Computing (ISAAC '13)*. Lecture Notes in Computer Science. to appear. Springer Verlag, 2013 (cited on p. 89).

[45] Kevin Buchin, Wouter Meulemans, and Bettina Speckmann. "A New Method for Subdivision Simplification with Applications to Urban-Area Generalization." In: *Proc. 19th ACM SIGSPATIAL International Conf. on Advances in Geographic Information Systems (GIS '11)*. ACM, 2011, pp. 261–270. DOI: 10.1145/2093973.2094009 (cited on p. 18).

[46] Sergio Cabello, Mark de Berg, Steven van Dijk, Marc van Kreveld, and Tycho Strijk. "Schematization of Road Networks." In: *Proc. 17th Ann. Symp. on Computational Geometry (SoCG '01)*. ACM, 2001, pp. 33–39. DOI: 10.1145/378583.378609 (cited on p. 17).

[47] Jason H. Cantarella, Eric D. Demaine, Hayley N. Iben, and James F. O'Brien. "An Energy-Driven Approach to Linkage Unfolding." In: *Proc. 20th Ann. Symp. on Computational Geometry (SoCG '04)*. 2004 (cited on p. 8).

[48] Dan Cartwright and Katie Atkinson. "Using Computational Argumentation to Support e-Participation." In: *IEEE Intelligent Systems* 24.5 (2009), pp. 42–52. DOI: 10.1109/MIS.2009.104 (cited on p. 56).

[49] Parinya Chalermsook and Julia Chuzhoy. "Maximum Independent Set of Rectangles." In: *Proc. 20th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA '09)*. 2009, pp. 892–901 (cited on p. 85).

[50] Timothy M. Chan. "A Note on Maximum Independent Sets in Rectangle Intersection Graphs." In: *Information Processing Letters* 89.1 (2004), pp. 19–23. DOI: 10.1016/j.ipl.2003.09.019 (cited on pp. 85, 86).

[51] Timothy M. Chan, Kasper Green Larsen, and Pătraşcu Mihai. "Orthogonal Range Searching on the RAM, Revisited." In: *Proc. 27th Ann. Symp. on Computational Geometry (SoCG '11)*. ACM, 2011, pp. 1–10. DOI: 10.1145/1998196.1998198 (cited on p. 11).

[52] Vijay Chandru and M. R. Rao. "Algorithms and Theory of Computation Handbook." In: Chapman & Hall/CRC, 2010. Chap. Linear Programming, pp. 30–38. URL: http://dl.acm.org/citation.cfm?id=1882757.1882787 (cited on p. 12).

[53] Bernard Chazelle and 36 co-authors. "The Computational Geometry Impact Task Force Report." In: *Advances in Discrete and Computional Geometry*. Vol. 223. American Mathematical Society, 1999, pp. 407–463 (cited on pp. 2, 81).

[54] Der-San Chen, Robert G. Batson, and Yu Dang. *Applied Integer Programming: Modeling and Solution*. Wiley, 2011 (cited on p. 155).

[55] Markus Chimani, Carsten Gutwenger, Petra Mutzel, and Hoi-Ming Wong. "Layer-Free Upward Crossing Minimization." In: *Journal of Experimental Algorithms* 15 (2010). DOI: 10.1145/1671970.1671975 (cited on pp. 55, 56, 59, 60, 76).

[56] Markus Chimani, Carsten Gutwenger, Petra Mutzel, and Hoi-Ming Wong. "Upward Planarization Layout." In: *Journal of Graph Algorithms and Applications* 15.1 (2011), pp. 127–155. DOI: 10.7155/jgaa.00220 (cited on pp. 55, 56).

[57] Jon Christensen, Stacy Friedman, Joe Marks, and Stuart Shieber. "Empirical Testing of Algorithms for Variable-Sized Label Placement." In: *Proc. 13th Ann. Symp. on Computational Geometry (SoCG '97)*. 1997, pp. 415–417 (cited on p. 87).

[58] Jon Christensen, Joe Marks, and Stuart Shieber. "Graphics Gems IV." In: Academic Press Professional, Inc., 1994. Chap. Placing Text Labels on Maps and Diagrams, pp. 497–504. URL: http://dl.acm.org/citation.cfm?id=180895.180942 (cited on p. 87).

[59] Jon Christensen, Joe Marks, and Stuart Shieber. "An Empirical Study of Algorithms for Point-Feature Label Placement." In: *ACM Transactions on Graphics* 14.3 (1995), pp. 203–232. DOI: 10.1145/212332.212334 (cited on p. 87).

[60] Stephen A. Cook. "The Complexity of Theorem-Proving Procedures." In: *Proc. 3rd Ann. ACM Symp. on Theory of Computing STOC '71*. STOC '71. ACM, 1971, pp. 151–158. DOI: 10.1145/800157.805047 (cited on p. 9).

[61] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. 3rd. The MIT Press, 2009 (cited on pp. 7, 8, 52).

[62] George B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963 (cited on p. 12).

[63] Daniel Delling, Andreas Gemsa, Martin Nöllenburg, and Thomas Pajor. "Path Schematization for Route Sketches." In: *Proc. 12th Scandinavian Symp. & Workshops on Algorithm Theory (SWAT '10)*. Vol. 6139. Lecture Notes in Computer Science. Springer Verlag, 2010, pp. 285–296. DOI: 10.1007/978-3-642-13731-0_27 (cited on pp. 3, 17, 18).

[64] Daniel Delling, Andreas Gemsa, Martin Nöllenburg, Thomas Pajor, and Ignaz Rutter. "On d-regular Schematization of Embedded Paths." In: *Computational Geometry: Theory and Applications* 47.3A (2014), pp. 381–406. DOI: 10.1016/j.comgeo.2013.10.002 (cited on pp. 3, 17).

[65] Giuseppe Di Battista, Walter Didimo, Maurizio Patrignani, and Maurizio Pizzonia. "Orthogonal and Quasi-upward Drawings with Vertices of Prescribed Size." In: *Proc. 7th International Symp. on Graph Drawing (GD '99)*. Vol. 1731. Lecture Notes in Computer Science. Springer Verlag, 1999, pp. 297–310. DOI: 10.1007/3-540-46648-7_31 (cited on p. 56).

[66] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. "Algorithms for Drawing Graphs: an Annotated Bibliography." In: *Computational Geometry: Theory and Applications* 4.5 (1994), pp. 235–282. DOI: 10.1016/0925-7721(94)00014-X (cited on p. 57).

[67] R. Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer Verlag, 2006 (cited on p. 7).

[68] Steven van Dijk, Dirk Thierens, and Mark de Berg. "Scalability and Efficiency of Genetic Algorithms for Geometrical Applications." In: *Proc. Parallel Problem Solving*

from Nature (PPSN VI). Vol. 1917. Lecture Notes in Computer Science. Springer Verlag, 2000, pp. 683–692. DOI: 10.1007/3-540-45356-3_67 (cited on p. 87).

[69] Srinivas Doddi, Madhav V. Marathe, Andy Mirzaian, Bernard M.E. Moret, and Binhai Zhu. "Map Labeling and its Generalizations." In: *Proc. 8th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA '97)*. 1997, pp. 148–157 (cited on pp. iv, 2, 81).

[70] Srinivas Doddi, Madhav V. Marathe, and Bernard M.E. Moret. "Point Set Labeling with Specified Positions." In: *International Journal of Computational Geometry & Applications* 12.1–2 (2002), pp. 29–66. DOI: 10.1142/S0218195902000736 (cited on p. 87).

[71] Christof Doll. "Automatic Layout Generation for Argument Maps." MA thesis. Karlsruhe Institute of Technology, 2012 (cited on pp. 4, 55, 69, 74).

[72] David H. Douglas and Thomas K. Peucker. "Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature." In: *Cartographica* 10.2 (1973), pp. 112–122. DOI: 10.3138/FM57-6770-U75U-7727 (cited on pp. 18, 46).

[73] Tim Dwyer, Yehuda Koren, and Kim Marriott. "Stress Majorization with Orthogonal Ordering Constraints." In: *Proc. 13th International Symp. on Graph Drawing (GD '05)*. Vol. 3843. Lecture Notes in Computer Science. Springer Verlag, 2006, pp. 141–152. DOI: 10.1007/11618058_14 (cited on p. 18).

[74] Tim Dwyer, Kim Marriott, and Michael Wybrow. "Interactive, Constraint-based Layout of Engineering Diagrams." In: *Electronic Communications of the EASST* 13 (2008) (cited on p. 58).

[75] Peter Eades. "A Heuristic for Graph Drawing." In: *Congressus Numerantium* 42 (1984), pp. 149–160 (cited on p. 57).

[76] Peter Eades, Xuemin Lin, and W. F. Smyth. "A Fast And Effective Heuristic For The Feedback Arc Set Problem." In: *Information Processing Letters* 47 (1993), pp. 319–323. DOI: 10.1016/0020-0190(93)90079-O (cited on p. 74).

[77] Peter Eades, Brendan D. McKay, and Nicholas C. Wormald. "On an Edge Crossing Problem." In: *Proc. 9th Australian Computational Science Conference*. Australian National University. 1986, pp. 327–334 (cited on p. 60).

[78] Dietmar Ebner, Gunnar W. Klau, and René Weiskircher. "Label Number Maximization in the Slider Model." In: *Proc. 12th International Symp. on Graph Drawing (GD '04)*. Vol. 3383. Lecture Notes in Computer Science. Springer Verlag, 2005, pp. 144–154. DOI: 10.1007/b105810 (cited on p. 87).

[79] Holger Eichelberger. "Aesthetics of Class Diagrams." In: *Visualizing Software for Understanding and Analysis*. 2002, pp. 23–31. DOI: 10.1109/VISSOF.2002.1019791 (cited on p. 58).

[80] Markus Eiglsperger, Carsten Gutwenger, Michael Kaufmann, Joachim Kupke, Michael Jünger, Sebastian Leipert, Karsten Klein, Petra Mutzel, and Martin Siebenhaller. "Automatic Layout of UML Class Diagrams in Orthogonal Style." In: *Information Visualization* 3.3 (2004), pp. 189–208. DOI: 10.1057/palgrave.ivs.9500078 (cited on p. 58).

[81] Peter van Emde Boas. "Machine Models and Simulation." In: *Handbook of Theoretical Computer Science*. Vol. A. Elsevier, 1990, pp. 1–66 (cited on p. 11).

[82] Thomas Erlebach, Torben Hagerup, Klaus Jansen, Moritz Minzlaff, and Alexander Wolff. "Trimming of Graphs, with Application to Point Labeling." In: *Theory of*

Computing Systems 47.3 (2010), pp. 613–636. DOI: 10.1007/s00224-009-9184-8 (cited on p. 87).

[83] esri. *MAPLEX—a Fully Automated Cartographic Name-Placement Software*. 1998. URL: http://www.esri.com/software/maplex/ (cited on p. 81).

[84] Jean-Daniel Fekete and Catharine Plaisant. "Excentric Labeling: Dynamic Neighborhood Labeling for Data Visualization." In: *Proc. Conference on Human Factors in Computer Systems (CHI '99)*. ACM, 1999, pp. 512–519. DOI: 10.1145/302979.303148 (cited on p. 89).

[85] Martin Fink, Jan-Henrik Haunert, André Schulz, Joachim Spoerhase, and Alexander Wolff. "Algorithms for Labeling Focus Regions." In: *IEEE Transactions on Visualization and Computer Graphics* 18.12 (2012), pp. 2583–2592. DOI: 10.1109/TVCG.2012.193 (cited on p. 143).

[86] Michael Formann and Frank Wagner. "A Packing Problem with Applications to Lettering of Maps." In: *Proc. 7th Ann. Symp. on Computational Geometry (SoCG '91)*. 1991, pp. 281–288. DOI: 10.1145/109648.109680 (cited on pp. 87, 95, 120, 121, 132, 142).

[87] Lance Fortnow. "The Status of the P versus NP Problem." In: *Communications of the ACM* 52.9 (2009), pp. 78–86. DOI: 10.1145/1562164.1562186 (cited on p. 8).

[88] Ulrich Fößmeier, Carsten Heß, and Michael Kaufmann. "On Improving Orthogonal Drawings: The 4M-Algorithm." In: *Proc. 6th International Symp. on Graph Drawing (GD '98)*. Vol. 1547. Lecture Notes in Computer Science. Springer Verlag, 1998, pp. 125–137. DOI: 10.1007/3-540-37623-2_10 (cited on pp. 69, 77).

[89] Robert J. Fowler, Michael S. Paterson, and Steven L. Tanimoto. "Optimal Packing and Covering in the Plane are NP-Complete." In: *Information Processing Letters* 12.3 (1981), pp. 133–137 (cited on p. 85).

[90] Peter Furlan. *Das Gelbe Rechenbuch: für Ingenieure, Naturwissenschaftler und Mathematiker*. Verlag Martina Furlan, 1995 (cited on p. 7).

[91] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979 (cited on pp. 8, 9, 10, 64, 149).

[92] Michael R. Garey and David S. Johnson. "Crossing Number is NP-complete." In: *SIAM Journal of Algebraic and Discrete Methods* 4.3 (1983), pp. 312–316. DOI: 10.1137/0604033 (cited on pp. 59, 60).

[93] Ashim Garg and Roberto Tamassia. "A New Minimum Cost Flow Algorithm with Applications to Graph Drawing." In: *Proc. 4th International Symp. on Graph Drawing (GD '96)*. Vol. 1190. Lecture Notes in Computer Science. Springer Verlag, 1997, pp. 201–216. DOI: 10.1007/3-540-62495-3_49 (cited on p. 69).

[94] Ashim Garg and Roberto Tamassia. "On the Computational Complexity of Upward and Rectilinear Planarity Testing." In: *SIAM Journal of Computing* 31.2 (2001), pp. 601–625. DOI: 10.1137/S0097539794277123 (cited on p. 59).

[95] Mari Garrido, Claudia Iturriaga, Alberto Márquez, José R. Portillo, Pedro Reyes, and Alexander Wolff. "Labeling Subway Lines." In: *Proc. 12th Ann. International Symp. on Algorithms and Computing (ISAAC '01)*. Vol. 2223. Lecture Notes in Computer Science. Springer Verlag, 2001, pp. 649–659. DOI: 10.1007/3-540-45678-3_55 (cited on pp. 149, 154, 166).

[96] Fanica Gavril. "The Intersection Graphs of Subtrees in Trees are Exactly the Chordal Graphs." In: *Journal of Combinatorial Theory. Series B* 16 (1974), pp. 47–56. DOI: 10.1016/0095-8956(74)90094-X (cited on p. 10).

[97] Andreas Gemsa. *Schematized Visualization of Shortest Paths in Road Networks*. Diploma thesis, Karlsruhe Institute of Technology, Germany, 2009. (cited on p. 17).

[98] Andreas Gemsa, Jan-Henrik Haunert, and Martin Nöllenburg. "Boundary-Labeling Algorithms for Panorama Images." In: *Proc. 19th ACM SIGSPATIAL International Conf. on Advances in Geographic Information Systems (SIGSPATIAL '11)*. 2011, pp. 289–298. DOI: 10.1145/2093973.2094012 (cited on p. 5).

[99] Andreas Gemsa, Jan-Henrik Haunert, and Martin Nöllenburg. "Multi-Row Boundary-Labeling Algorithms for Panorama Images." In: *ACM Transactions on Spatial Algorithms and Systems* (2014). accepted for publication (cited on p. 5).

[100] Andreas Gemsa, Benjamin Niedermann, and Martin Nöllenburg. "Trajectory-Based Dynamic Map Labeling." In: *Proc. 24th Ann. International Symp. on Algorithms and Computing (ISAAC '13)*. Vol. 8283. Lecture Notes in Computer Science. Full version available at http://arxiv.org/abs/1309.3963. Springer Verlag, 2013. DOI: 10.1007/978-3-642-45030-3_39 (cited on pp. 89, 95, 111, 112).

[101] Andreas Gemsa, Martin Nöllenburg, Thomas Pajor, and Ignaz Rutter. *On d-regular Schematization of Embedded Paths*. Tech. rep. 2010-21. Faculty of Informatics, Karlsruhe Institute of Technology, 2010. URL: http://digbib.ubka.uni-karlsruhe.de/volltexte/1000020426 (cited on pp. 3, 17).

[102] Andreas Gemsa, Martin Nöllenburg, Thomas Pajor, and Ignaz Rutter. "On d-regular Schematization of Embedded Paths." In: *Proc. 37th International Conf. on Current Trends in Theory and Practice of Computer Science (SOFSEM '11)*. Vol. 6543. Lecture Notes in Computer Science. Springer Verlag, 2011, pp. 260–271. DOI: 10.1007/978-3-642-18381-2_22 (cited on pp. 3, 17).

[103] Andreas Gemsa, Martin Nöllenburg, and Ignaz Rutter. "Consistent Labeling of Rotating Maps." In: *Proceedings of the 27th European Workshop on Computational Geometry (EuroCG '11)*. 2011, pp. 171–174 (cited on pp. 4, 93).

[104] Andreas Gemsa, Martin Nöllenburg, and Ignaz Rutter. "Consistent Labeling of Rotating Maps." In: *Proc. 12th International Workshop on Algorithms and Data Structures (WADS '11)*. Vol. 6844. Lecture Notes in Computer Science. Full version available at http://arxiv.org/abs/1104.5634. Springer Verlag, 2011, pp. 451–462. URL: http://dx.doi.org/10.1007/978-3-642-22300-6_38 (cited on pp. 4, 88, 93, 106, 132).

[105] Andreas Gemsa, Martin Nöllenburg, and Ignaz Rutter. "Sliding Labels for Dynamic Point Labeling." In: *Proc. 23rd Canadian Conference on Computional Geometry (CCCG '11)*. University of Toronto Press, 2011 (cited on pp. 5, 131).

[106] Andreas Gemsa, Martin Nöllenburg, and Ignaz Rutter. "Evaluation of Labeling Strategies for Rotating Maps." In: *Proc. 13th International Symp. on Experimental Algorithms (SEA '14)*. Lecture Notes in Computer Science. Full, version available at http://arxiv.org/abs/1404.1849. Springer Verlag, 2014, pp. 235–246. DOI: 10.1007/978-3-319-07959-2_20 (cited on pp. 4, 93).

[107] Eric Gervais, Doron Nussbaum, and Jörg-Rüdiger Sack. "DynaMap: A Context Aware Dynamic Map Application." In: *Proc. GISPlanet*. 2005 (cited on p. 93).

[108] Freeman H., Marrinan S., and Chitalia H. "Automated Labeling of Soil Survey Maps." In: *ASPRS-ACSM Annual Convention*. Vol. 1. 1996, pp. 51–59 (cited on p. 89).

[109] Leonid G. Haĉijan. "A Polynomial Algorithm in Linear Programming." In: *Soviet Mathematics Doklady* 20 (1979), pp. 191–194 (cited on p. 12).

[110] Hsu-Chen Yen Hao-Jen Kao Chun-Cheng Lin. "Many-to-One Boundary Labeling." In: *Proc. Asia-Pacific Symposium on Visualization (APVIS '07)*. IEEE Computer Society, 2007, pp. 65–72. DOI: 10.7155/jgaa.00169 (cited on p. 143).

[111] Knut Hartmann, Timo Götzelmann, Kamran Ali, and Thomas Strothotte. "Metrics for Functional and Aesthetic Label Layouts." In: *Proc. 5th International Conference on Smart Graphics (SG '05)*. Vol. 3638. Lecture Notes in Computer Science. Springer Verlag, 2005, pp. 115–126. DOI: 10.1007/11536482_10 (cited on p. 142).

[112] Johan T. Håstad. "Clique is Hard to Approximate within $n^{1-\varepsilon}$." In: *Proc. 37th Ann. IEEE Symp. on Foundations of Computer Science (FOCS '96)*. IEEE Computer Society, 1996, p. 627. URL: http://dl.acm.org/citation.cfm?id=874062.875499 (cited on p. 85).

[113] Kunihiko Hayashi, Michiko Inoue, Toshimitsu Masuzawa, and Hideo Fujiwara. "A Layout Adjustment Problem for Disjoint Rectangles Preserving Orthogonal Order." In: *Proc. 6th International Symp. on Graph Drawing (GD '98)*. Vol. 1547. Lecture Notes in Computer Science. Springer Verlag, 1998, pp. 183–197. DOI: 10.1002/scj.1104 (cited on p. 18).

[114] Weiqing He and Kim Marriott. "Constrained Graph Layout." In: *Proc. 5th International Symp. on Graph Drawing (GD '97)*. Vol. 1190. Lecture Notes in Computer Science. Springer Verlag, 1997, pp. 217–232. DOI: 10.1007/3-540-62495-3_50 (cited on p. 57).

[115] Frank Heidmann, Fabian Hermann, and Matthias Peissner. "Interactive Maps on Mobile, Location-Based Systems: Design Solutions and Usability Testing." In: *Proc. 21st International Cartographic Conference*. 2003, pp. 10–16 (cited on p. 88).

[116] Stephen A. Hirsch. "An Algorithm for Automated Placement of Point Data." MA thesis. Department of Geography, State University of New York at Buffalo, New York, 1980 (cited on p. 87).

[117] Dorit S. Hochbaum and Wolfgang Maass. "Approximation Schemes for Covering and Packing Problems in Image Processing and VLSI." In: *Journal of the ACM* 32.1 (1985), pp. 130–136. DOI: 10.1145/2455.214106 (cited on pp. 4, 86, 107).

[118] Hartwig Hochmair. "The Influence of Map Design on Route Choice from Public Transportation Maps in Urban Areas." In: *The Cartographic Journal* 46.3 (2009), pp. 242–256. DOI: 10.1179/000870409X12472347560623 (cited on p. 1).

[119] Weidong Huang, Seok-Hee Hong, and P. Eades. "Effects of Crossing Angles." In: *IEEE Pacific Vis. Symp. PacificVis 2008*. 2008, pp. 41–46. DOI: 10.1109/PACIFICVIS.2008.4475457 (cited on p. 56).

[120] Zhi-Dong Huang, Sheung-Hung Poon, and Chun-Cheng Lin. "Boundary Labeling with Flexible Label Positions." In: *Proc. 8th International Workshop on Algorithms and Computation (WALCOM '14)*. Vol. 8344. Lecture Notes in Computer Science. Springer Verlag, 2014, pp. 44–55. DOI: 10.1007/978-3-319-04657-0_7 (cited on p. 143).

[121] Hiroshi Imai and Takao Asano. "Finding the Connected Components and a Maximum Clique of an Intersection Graph of Rectangles in the Plane." In: *Journal of Algorithms* 4 (1983), pp. 310–323 (cited on p. 85).

[122] Eduard Imhof. "Die Anordnung der Namen in der Karte." German. In: *Internat. Yearbook of Cartography*. Kirschbaum, 1962, pp. 93–129 (cited on p. 82).

[123] Eduard Imhof. "Positioning Names on Maps." In: *The American Cartographer* 2.2 (1975), pp. 128–144 (cited on pp. iv, 2, 81, 82, 131, 133, 134).

[124] David Johnson. "The Many Faces of Polynomial Time." In: *Journal of Algorithms* 8.2 (1987), pp. 285–303 (cited on p. 8).

[125] Michael Jünger and Petra Mutzel. *2-Layer Straightline Crossing Minimization: Performance of Exact And Heuristic Algorithms*. MPI Informatik, Bibliothek & Dokumentation, 1996 (cited on p. 60).

[126] Narendra K. Karmakar. "A New Polynomial-Time Algorithm for Linear Programming." In: *Combinatorica* 4.1 (1984) (cited on p. 12).

[127] Richard M. Karp. "Reducibility Among Combinatorial Problems." In: *Complexity of Computer Computations*. Plenum Press, 1972, pp. 85–103. DOI: 10.1007/978-1-4684-2001-2_9 (cited on pp. 9, 10, 12, 74).

[128] Michael Kaufmann. "On Map Labeling with Leaders." In: *Festschrift Mehlhorn*. Vol. 5760. Lecture Notes in Computer Science. Springer Verlag, 2009, pp. 290–304. DOI: 10.1007/978-3-642-03456-5_20 (cited on p. 142).

[129] Philipp Kindermann, Benjamin Niedermann, Ignaz Rutter, Marcus Schaefer, André Schulz, and Alexander Wolff. "Two-Sided Boundary Labeling with Adjacent Sides." In: *Proc. 13th International Workshop on Algorithms and Data Structures (WADS '13)*. Vol. 8037. Lecture Notes in Computer Science. Springer Verlag, 2013, pp. 463–474. DOI: 10.1007/978-3-642-40104-6_40 (cited on p. 143).

[130] Gunnar W. Klau. "A Combinatorial Approach to Orthogonal Placement Problems." PhD thesis. Universität des Saarlandes, 2001. URL: http://scidok.sulb.uni-saarland.de/volltexte/2004/196/pdf/GunnarWernerKlau_ProfDrKurtMehlhorn.pdf (cited on p. 87).

[131] Gunnar W. Klau and Petra Mutzel. "Optimal Labeling of Point Features in Rectangular Labeling Models." In: *Mathematical Programming (Series B)* (2003), pp. 435–458. DOI: 10.1007/s10107-002-0327-9 (cited on pp. 95, 132).

[132] Victor Klee and George J. Minty. "How Good is the Simplex Algorithm?" In: *Inequalities, III (Proc. Third Sympos., Univ. California, Los Angeles, Calif., 1969; dedicated to the memory of Theodore S. Motzkin)*. Academic Press, 1972, pp. 159–175 (cited on p. 12).

[133] Alexander Klippel, Kai-Florian Richter, Thomas Barkowsky, and Christian Freksa. "The Cognitive Reality of Schematic Maps." In: *Map-Based Mobile Services*. Springer Verlag, 2005, pp. 55–71. DOI: 10.1007/3-540-26982-7_5 (cited on p. 1).

[134] Johannes Kopf, Maneesh Agrawala, David Bargeron, David Salesin, and Michael Cohen. "Automatic Generation of Destination Maps." In: *ACM Transactions on Graphics* 29.6 (2010), 158:1–158:12. DOI: 10.1145/1882261.1866184 (cited on p. 18).

[135] Eric R. Kramers. "Interaction with Maps on the Internet - A User Centred Design Approach for The Atlas of Canada." In: *The Cartographic Journal* 45.2 (2008), pp. 98–107. DOI: 10.1179/174327708X305094 (cited on p. 88).

[136] Marc van Kreveld, Tycho Strijk, and Alexander Wolff. "Point Labeling with Sliding Labels." In: *Computational Geometry: Theory and Applications* 13 (1999), pp. 21–47. DOI: 10.1016/S0925-7721(99)00005-X (cited on pp. 83, 87, 95, 107, 132, 140, 142).

[137] Melvin R. Krom. "The Decision Problem for a Class of First-Order Formulas in Which all Disjunctions are Binary." In: *Mathematical Logic Quarterly* 13.1–2 (1967), pp. 15–20. DOI: 10.1002/malq.19670130104 (cited on pp. 10, 87).

[138] D. H. Lehmer. “A Note on Trigonometric Algebraic Numbers.” In: *The American Mathematical Monthly* 40.3 (1933), pp. 165–166. DOI: 10.2307/2301023 (cited on pp. 32, 38).

[139] David Lichtenstein. “Planar Formulae and Their Uses.” In: *SIAM J. Comput.* 11.2 (1982), pp. 329–343. DOI: 10.1137/0211025 (cited on pp. 10, 100).

[140] Chun-Cheng Lin. “Crossing-Free Many-to-One Boundary Labeling with Hyperleaders.” In: *Proc. IEEE Pacific Visualisation Symposium (PacificVis’10)*. 2010, pp. 185–192. DOI: 10.1109/PACIFICVIS.2010.5429592 (cited on p. 143).

[141] Joseph Marks and Stuart M Shieber. *The Computational Complexity of Cartographic Label Placement*. 1991 (cited on p. 85).

[142] Rafael Marti and Manuel Laguna. “Heuristics and Meta-Heuristics for 2-Layer Straight Line Crossing Minimization.” In: *Discrete Applied Mathematics* 127.3 (2003), pp. 665–678. DOI: 10.1016/S0166-218X(02)00397-9 (cited on p. 60).

[143] Damian Merrick and Joachim Gudmundsson. “Path Simplification for Metro Map Layout.” In: *Proc. 14th International Symp. on Graph Drawing (GD ’06)*. Vol. 4372. Lecture Notes in Computer Science. Springer Verlag, 2007, pp. 258–269. DOI: 10.1007/978-3-540-70904-6_26 (cited on p. 18).

[144] Kazuo Misue, Peter Eades, Wei Lai, and Kozo Sugiyama. “Layout Adjustment and the Mental Map.” In: *Journal of Visual Languages and Computing* 6.2 (1995), pp. 183–210. DOI: 10.1006/jvlc.1995.1010 (cited on p. 17).

[145] Mark S. Monmonier. *How to Lie with Maps*. University of Chicago Press, 1996 (cited on p. 1).

[146] Christian Worm Mortensen. “Fully-Dynamic Two Dimensional Orthogonal Range and Line Segment Intersection Reporting in Logarithmic Time.” In: *Proc. 14th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA ’03)*. SODA ’03. Society for Industrial and Applied Mathematics, 2003, pp. 618–627. URL: http://dl.acm.org/citation.cfm?id=644108.644210 (cited on p. 42).

[147] Kevin D. Mote. “Fast Point-Feature Label Placement for Dynamic Visualizations.” In: *Information Visualization* 6.4 (2007), pp. 249–260. DOI: 10.1057/palgrave.ivs.9500163 (cited on pp. 95, 132).

[148] Gabriele Neyer. “Line Simplification with Restricted Orientations.” In: *Proc. 6th International Workshop on Algorithms and Data Structures (WADS ’99)*. Vol. 1663. Lecture Notes in Computer Science. Springer Verlag, 1999, pp. 13–24. DOI: 10.1007/3-540-48447-7_2 (cited on p. 18).

[149] Annu-Maaria Nivala, Stephen Brewster, and Tiina Sarjakoski. “Usability Evaluation of Web Mapping Sites.” In: *The Cartographic Journal* 45.2 (2008), pp. 129–138. DOI: 10.1002/9780470979587.ch49 (cited on p. 88).

[150] Martin Nöllenburg. *Automated Drawing of Metro Maps*. Tech. rep. 2005-25. Fakultät für Informatik, Universität Karlsruhe, 2005. URL: http://digbib.ubka.uni-karlsruhe.de/volltexte/1000004123 (cited on p. 18).

[151] Martin Nöllenburg, Valentin Polishchuk, and Mikko Sysikaski. “Dynamic One-Sided Boundary Labeling.” In: *Proc. 18th ACM SIGSPATIAL International Conf. on Advances in Geographic Information Systems (ACM GIS ’10)*. 2010, pp. 310–319. DOI: 10.1145/1869790.1869834 (cited on pp. 93, 95, 132).

[152] Martin Nöllenburg, Valentin Polishchuk, and Mikko Sysikaski. “Dynamic One-Sided Boundary Labeling.” In: *Proc. 18th ACM SIGSPATIAL International Conf. on*

Advances in Geographic Information Systems (ACM GIS 2010). 2010, pp. 310–319. DOI: 10.1145/1869790.1869834 (cited on p. 143).

[153] Martin Nöllenburg and Alexander Wolff. "Drawing and Labeling High-Quality Metro Maps by Mixed-Integer Programming." In: *IEEE Transactions on Visualization and Computer Graphics* 17.5 (2011), pp. 626–641. DOI: 10.1109/TVCG.2010.81 (cited on pp. 15, 18, 43).

[154] Kristien Ooms, Wim Kellens, and Veerle Fack. "Dynamic Map Labeling for Users." In: *Proc. 24th International Cartographic Conference (ICC '09)*. 2009 (cited on pp. 88, 131).

[155] Christos H. Papadimitriou. *Computational Complexity*. 12th ed. Addison-Wesley, 2005 (cited on p. 8).

[156] Ingo Petzold, Gerhard Gröger, and Lutz Plümer. "Fast Screen Map Labeling—Data-Structures and Algorithms." In: *Proc. 23rd International Cartographic Conference (ICC '03)*. 2003, pp. 288–298 (cited on pp. 88, 95, 132).

[157] Ingo Petzold, Gerhard Gröger, and Lutz Plümer. "Modeling of Conflicts for Screen Map Labeling." In: *Proc. 20th ISPRS Congress*. Vol. 34, Part B4. Internat. Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. 2004 (cited on p. 88).

[158] Ingo Petzold, Lutz Plümer, and Markus Heber. "Label Placement for Dynamically Generated Screen Maps." In: *Proc. 19th International Cartographic Conference (ICC '99)*. 1999, pp. 893–903. URL: http://www.ikg.uni-bonn.de/Publikationen/ica99-paper%20pdf-ps.zip (cited on p. 88).

[159] Sheung-Hung Poon and Chan-Su Shin. "Adaptive Zooming in Point Set Labeling." In: *Proc. 15th Fundamentals on Computational Theory (FCT '05)*. Vol. 3623. Lecture Notes in Computer Science. 2005, pp. 233–244. DOI: 10.1007/11537311_21 (cited on pp. 88, 132).

[160] Sheung-Hung Poon, Chan-Su Shin, Tycho Strijk, Takeaki Uno, and Alexander Wolff. "Labeling Points with Weights." In: *Algorithmica* 38.2 (2003), pp. 341–362. DOI: 10.1007/s00453-003-1063-0 (cited on pp. 86, 146).

[161] Andreas Reimer and Maxim Rylov. "Point-feature Lettering of High Cartographic Quality: A Multi-criteria Model with Practical Implementation." In: *Proc. 30th European Workshop on Computational Geometry (EuroCG '14)*. 2014 (cited on pp. 84, 97).

[162] Glaydston Mattos Ribeiro and Luiz Antonio Nogueira Lorena. "Heuristics for Cartographic Label Placement Problems." In: *Computational Geoscience* 32.6 (2006), pp. 739–748. DOI: 10.1016/j.cageo.2005.10.004 (cited on p. 87).

[163] Neil Robertson and Paul D. Seymour. "Graph Minors .XIII. The Disjoint Paths Problem." In: *Journal of Combinatorial Theory, Series B* 63.1 (1995), pp. 65–110. DOI: 10.1006/jctb.1995.1006 (cited on p. 8).

[164] Sasanka Roy, Subhasis Bhattacharjee, Sandip Das, and Subhas C. Nandy. "A New Fast Heuristic for Labeling Points." In: *Information Processing Letters* 109.10 (2009), pp. 478–484. DOI: 10.1016/j.ipl.2009.01.011 (cited on p. 87).

[165] Peter Sanders and Dorothea Wagner. "Algorithm Engineering." In: *it - Information Technology* 53.6 (2011), pp. 263–265. DOI: http://dx.doi.org/10.1524/itit.2011.9072 (cited on p. 171).

[166] David C. Schneider, Christian Voigt, and Gregor Betz. "ArguNet—a Software Tool for Collaborative Argumentation Analysis and Research." In: *7th Workshop on Computational Models of Natural Argument (CMNA VII)*. 2007 (cited on pp. 55, 74).

[167] Arnold Schönhage. "On the Power of Random Access Machines." In: *Automata, Languages and Programming*. Vol. 71. Lecture Notes in Computer Science. Springer Verlag, 1979, pp. 520–529. DOI: 10.1007/3-540-09510-1_42 (cited on p. 11).

[168] Falk Schreiber, Tim Dwyer, Kim Marriott, and Michael Wybrow. "A Generic Algorithm for Layout of Biological Networks." In: *BMC Bioinformatics* 10.1 (2009), p. 375. DOI: 10.1186/1471-2105-10-375 (cited on p. 58).

[169] Jochen Seemann. "Extending the Sugiyama Algorithm for Drawing UML Class Diagrams: Towards Automatic Layout of Object-Oriented Software Diagrams." In: *Proc. 5th International Symp. on Graph Drawing (GD '97)*. Vol. 1353. Lecture Notes in Computer Science. Springer Verlag, 1997, pp. 415–424. DOI: 10.1007/3-540-63938-1_86 (cited on p. 58).

[170] N.Z. Shor. "Convergence Rate of the Gradient Descent Method with Dilatation of the Space." In: *Cybernetics* 6.2 (1970), pp. 102–108. DOI: 10.1007/BF01070506 (cited on p. 12).

[171] Janet M. Six, Konstantinos G. Kakoulis, and Ioannis G. Tollis. "Refinement of Orthogonal Graph Drawings." In: *Proc. 6th International Symp. on Graph Drawing (GD '98)*. Vol. 1547. Lecture Notes in Computer Science. Springer Verlag, 1998, pp. 302–315. DOI: 10.1007/3-540-37623-2_23 (cited on p. 68).

[172] John P. Snyder. *Flattening the Earth: Two Thousand Years of Map Projections*. University of Chicago Press, 1993 (cited on p. 121).

[173] Bettina Speckmann and Kevin Verbeek. "Homotopic Rectilinear Routing with Few Links and Thick Edges." In: *LATIN 2010: Theoretical Informatics*. Vol. 6034. Lecture Notes in Computer Science. Springer Verlag, 2010, pp. 468–479. DOI: 10.1007/978-3-642-12200-2_41 (cited on p. 18).

[174] Tycho Strijk and Marc van Kreveld. "Practical Extensions of Point Labeling in the Slider Model." In: *GeoInformatica* 6.2 (2002), pp. 181–197 (cited on p. 132).

[175] Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. "Methods for Visual Understanding of Hierarchical System Structures." In: *IEEE Transactions on Systems, Man and Cybernetics* 11.2 (1981), pp. 109–125. DOI: 10.1109/TSMC.1981.4308636 (cited on pp. 16, 55).

[176] Elijah Swift. "Discussions: Note on Trigonometric Functions." In: *The American Mathematical Monthly* 29.10 (1922), pp. 404–405. DOI: 10.2307/2299031 (cited on p. 38).

[177] Roberto Tamassia. "On Embedding a Graph in the Grid with the Minimum Number of Bends." In: *SIAM Journal of Computing* 16 (3 1987), pp. 421–444. DOI: 10.1137/0216030 (cited on pp. 16, 55, 56, 69).

[178] P.M. Vaidya. "Speeding-up Linear Programming Using Fast Matrix Multiplication." In: *Proc. 30th Ann. IEEE Symp. on Foundations of Computer Science (FOCS '89)*. 1989, pp. 332–337. DOI: 10.1109/SFCS.1989.63499 (cited on pp. 27, 38).

[179] Vijay V. Vazirani. *Approximation Algorithms*. Springer Verlag, 2001 (cited on p. 12).

[180] Kevin Verbeek. "Homotopic $\mathcal{C}$-Oriented Routing." In: *Proc. 20th International Symp. on Graph Drawing (GD '12)*. Lecture Notes in Computer Science. Springer Verlag, 2013, pp. 272–278. DOI: 10.1007/978-3-642-36763-2_24 (cited on p. 18).

[181] Frank Wagner and Alexander Wolff. "A Practical Map Labeling Algorithm." In: *Computational Geometry: Theory and Applications* 7 (1997), pp. 387–404 (cited on p. 132).

[182] Frank Wagner, Alexander Wolff, Vikas Kapoor, and Tycho Strijk. "Three Rules Suffice for Good Label Placement." In: *Algorithmica* 30.2 (2001), pp. 334–349. DOI: 10.1007/s00453-001-0009-7 (cited on pp. 95, 132).

[183] Ingo Wegener. *Complexity Theory - Exploring the Limits of Efficient Algorithms*. Springer Verlag, 2005, pp. I–XI, 1–308. DOI: 10.1007/3-540-27477-4 (cited on p. 8).

[184] John Noble Wilford. *The Mapmakers*. Vintage Series. Vintage Books, 2001 (cited on p. 1).

[185] Alexander Wolff. "Drawing Subway Maps: A Survey." In: *Informatik - Forschung und Entwicklung* 22.1 (2007), pp. 23–44. DOI: 10.1007/s00450-007-0036-y (cited on p. 18).

[186] Alexander Wolff and Tycho Strijk. *The Map-Labeling Bibliography*. 1996. URL: http://i11www.iti.kit.edu/map-labeling/bibliography (cited on pp. 84, 95, 132, 142).

[187] Missae Yamamoto, Gilberto Camara, and Luiz Antonio Nogueira Lorena. "Tabu Search Heuristic for Point-Feature Cartographic Label Placement." In: *GeoInformatica* 6.1 (2002), pp. 77–90. DOI: 10.1023/A:1013720231747 (cited on p. 87).

[188] Pinhas Yoeli. "The Logic of Automated Map Lettering." In: *The Cartographic Journal* 9.2 (1972), pp. 99–108. DOI: 10.1179/000870472787352451 (cited on p. 82).

[189] Yusuke Yokosuka and Keiko Imai. "Polynomial Time Algorithms for Label Size Maximization on Rotating Maps." In: *CCCG 2013*. Proc. 25th Canadian Conference on Computional Geometry (CCCG '13). 2013, pp. 187–192 (cited on pp. 88, 95).

[190] Steven Zoraster. "Practical Results Using Simulated Annealing for Point Feature Label Placement." In: *Cartography & Geographic Information Science* 24.4 (1997), pp. 228–238. DOI: 10.1559/152304097782439259 (cited on p. 89).

LIST OF PUBLICATIONS

JOURNAL ARTICLES

[1] Gregor Betz, Andreas Gemsa, Christof Mathies, Ignaz Rutter, and Dorothea Wagner. "Column-based Graph Layouts." In: *Journal of Graph Algorithms and Applications* 18.5 (2014), pp. 677–708. DOI: 10.7155/jgaa.00341.

[2] Edith Brunel, Andreas Gemsa, Marcus Krug, Ignaz Rutter, and Dorothea Wagner. "Generalizing Geometric Graphs." In: *Journal of Graph Algorithms and Applications* 18.1 (2014), pp. 35–76. DOI: 10.7155/jgaa.00314.

[3] Daniel Delling, Andreas Gemsa, Martin Nöllenburg, Thomas Pajor, and Ignaz Rutter. "On d-regular Schematization of Embedded Paths." In: *Computational Geometry: Theory and Applications* 47.3A (2014), pp. 381–406. DOI: 10.1016/j.comgeo.2013.10.002.

[4] Andreas Gemsa, Jan-Henrik Haunert, and Martin Nöllenburg. "Multi-Row Boundary-Labeling Algorithms for Panorama Images." In: *ACM Transactions on Spatial Algorithms and Systems* (2014). accepted for publication.

CONFERENCE ARTICLES

[1] Andreas Gemsa, Benjamin Niedermann, and Martin Nöllenburg. "Label Placement in Road Maps." In: *Proc. 29th European Workshop on Computational Geometry (EuroCG '14)*. 2014.

[2] Andreas Gemsa, Martin Nöllenburg, and Ignaz Rutter. "Evaluation of Labeling Strategies for Rotating Maps." In: *Proc. 13th International Symp. on Experimental Algorithms (SEA '14)*. Lecture Notes in Computer Science. Full, version available at http://arxiv.org/abs/1404.1849. Springer Verlag, 2014, pp. 235–246. DOI: 10.1007/978-3-319-07959-2_20.

[3] Gregor Betz, Christof Doll, Andreas Gemsa, Ignaz Rutter, and Dorothea Wagner. "Column-based Graph Layouts." In: *Proc. 20th International Symp. on Graph Drawing (GD '12)*. Vol. 7704. Lecture Notes in Computer Science. Springer Verlag, 2013, pp. 236–247. DOI: 10.1007/978-3-642-36763-2_21.

[4] Andreas Gemsa, Benjamin Niedermann, and Martin Nöllenburg. "Trajectory-Based Dynamic Map Labeling." In: *Proc. 29th European Workshop on Computational Geometry (EuroCG '13)*. 2013.

[5] Andreas Gemsa, Benjamin Niedermann, and Martin Nöllenburg. "Trajectory-Based Dynamic Map Labeling." In: *Proc. 24th Ann. International Symp. on Algorithms and Computing (ISAAC '13)*. Vol. 8283. Lecture Notes in Computer Science. Full version available at http://arxiv.org/abs/1309.3963. Springer Verlag, 2013. DOI: 10.1007/978-3-642-45030-3_39.

[6] Andreas Gemsa, Thomas Pajor, Dorothea Wagner, and Tobias Zündorf. "Efficient Computation of Jogging Routes." In: *Proc. 12th International Symp. on Experimental Algorithms (SEA '13)*. Vol. 7933. Lecture Notes in Computer Science. Springer Verlag, 2013, pp. 272–283.

[7] Edith Brunel, Andreas Gemsa, Marcus Krug, Ignaz Rutter, and Dorothea Wagner. "Generalizing Geometric Graphs." In: *Proc. 19th International Symp. on Graph Drawing (GD '11)*. Lecture Notes in Computer Science. Springer Verlag, 2012, pp. 179–190. DOI: 10.1007/978-3-642-25878-7_18.

[8] Andreas Gemsa, D.T. Lee, Chih-Hung Liu, and Dorothea Wagner. "Higher Order City Voronoi Diagrams." In: *Proc. 28th European Workshop on Computational Geometry (EuroCG '12)*. 2012, pp. 245–248.

[9] Andreas Gemsa, D.T. Lee, Chih-Hung Liu, and Dorothea Wagner. "Higher Order City Voronoi Diagrams." In: *Proc. 13th Scandinavian Symp. & Workshops on Algorithm Theory (SWAT '12)*. Vol. 7357. Lecture Notes in Computer Science. Springer Verlag, 2012, pp. 59–70. DOI: 10.1007/978-3-642-31155-0_6.

[10] Andreas Gemsa, Jan-Henrik Haunert, and Martin Nöllenburg. "Boundary-Labeling Algorithms for Panorama Images." In: *Proc. 19th ACM SIGSPATIAL International Conf. on Advances in Geographic Information Systems*. ACM Press, 2011, pp. 289–298. DOI: 10.1145/2093973.2094012.

[11] Andreas Gemsa, Martin Nöllenburg, Thomas Pajor, and Ignaz Rutter. "Automatic Generation of Route Sketches." In: *Proc. 18th International Symp. on Graph Drawing (GD '10)*. Vol. 6502. Lecture Notes in Computer Science. Poster abstract. Springer Verlag, 2011, pp. 391–392. DOI: 10.1007/978-3-642-18469-7_37.

[12] Andreas Gemsa, Martin Nöllenburg, Thomas Pajor, and Ignaz Rutter. "On d-regular Schematization of Embedded Paths." In: *Proc. 37th International Conf. on Current Trends in Theory and Practice of Computer Science (SOFSEM '11)*. Vol. 6543. Lecture Notes in Computer Science. Springer Verlag, 2011, pp. 260–271. DOI: 10.1007/978-3-642-18381-2_22.

[13] Andreas Gemsa, Martin Nöllenburg, and Ignaz Rutter. "Consistent Labeling of Rotating Maps." In: *Proc. 27th European Workshop on Computational Geometry (EuroCG '11)*. 2011, pp. 171–174.

[14] Andreas Gemsa, Martin Nöllenburg, and Ignaz Rutter. "Consistent Labeling of Rotating Maps." In: *Proc. 12th International Workshop on Algorithms and Data Structures (WADS '11)*. Vol. 6844. Lecture Notes in Computer Science. Full version available at http://arxiv.org/abs/1104.5634. Springer Verlag, 2011, pp. 451–462. DOI: 10.1007/978-3-642-22300-6_38.

[15] Andreas Gemsa, Martin Nöllenburg, and Ignaz Rutter. "Sliding Labels for Dynamic Point Labeling." In: *Proc. 23rd Canadian Conference on Computional Geometry (CCCG '11)*. University of Toronto Press, 2011.

[16] Edith Brunel, Daniel Delling, Andreas Gemsa, and Dorothea Wagner. "Space-Efficient SHARC-Routing." In: *Proc. 9th International Symp. on Experimental Algorithms (SEA '10)*. Vol. 6049. Lecture Notes in Computer Science. Springer Verlag, 2010, pp. 47–58. DOI: 10.1007/978-3-642-13193-6_5.

[17] Daniel Delling, Andreas Gemsa, Martin Nöllenburg, and Thomas Pajor. "Path Schematization for Route Sketches." In: *Proc. 12th Scandinavian Symp. & Workshops on Algorithm Theory (SWAT '10)*. Vol. 6139. Lecture Notes in Computer Science. Springer Verlag, 2010, pp. 285–296. DOI: 10.1007/978-3-642-13731-0_27.

ERKLÄRUNG

Ich versichere, diese Dissertation selbstständig angefertigt, alle benutzten Hilfsmittel vollständig angegeben haben. Außerdem versichere ich, dass kenntlich gemacht zu haben, was aus Arbeiten anderer und eigener Veröffentlichungen unverändert oder mit Änderungen entnommen wurde.

Karlsruhe, 14.02.2015

Andreas Gemsa