Network Visualization:

Algorithms, Applications, and Complexity

zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften

der Fakultät für Informatik
der Universität Fridericiana zu Karlsruhe (TH)

genehmigte
Dissertation
von
Martin Nöllenburg
aus Heilbronn

Tag der mündlichen Prüfung:	13. Februar 2009
Erster Gutachter:	Herr PD Dr. Alexander Wolff
Zweiter Gutachter:	Frau Prof. Dr. Dorothea Wagner

Für Otto.

Acknowledgments

Pursuing my PhD has been a demanding, but also very exciting and enjoyable project culminating in this thesis after almost three and a half years of work. It all started in one of those moments in life, where you run into an opportunity and—without too much pondering—you seize it. This was the case when Sascha offered me a position as a PhD student in his junior research group GeoNet in 2005 after I had just written my diploma thesis with him. Today, I can say with conviction that accepting his offer and committing myself to work on a research project in the intersection of algorithmics, computational geometry, and visualization was the right decision.

I am grateful to many people who directly or indirectly contributed to this thesis. In the first place, I want to mention my advisor, teacher, and mentor Alexander (Sascha) Wolff. It is a simple statement: without you I would never have written this thesis. You have taken your role as an advisor seriously. Your door was always open for my concerns, and even after our "breakup" between Karlsruhe and Eindhoven you were available for discussions by mail and phone (almost) anytime. I have learned not only from your technical knowledge, but I have also been (and still am) thankfully learning two very fundamental skills for working in research and science: concise academic writing and creating appealing presentations. Moreover, you were steadily encouraging me to go to international conferences, workshops, and summer schools, which is crucial to start becoming part of the scientific community. What more can a PhD student hope for? Thank you for all your support, Sascha!

I also want to thank Dorothea Wagner for "adopting" me in her group after Sascha left for Eindhoven and for agreeing to be my second thesis reviewer. I further thank my office mates and fellow "GeoNetlers" Marc Benkert and Ignaz Rutter for the great time we had over the last years and for always being the first persons to ask for all kinds of advice. Let me extend this acknowledgment to all my colleagues and friends in the Algo group and to our invaluable secretary Lilian Beckert for making the institute a great place to work. Whatever the reason, there was always someone to have a coffee and a chat with.

With respect to the contents of my thesis I want to thank my numerous co-authors for their ideas and contributions to our joint work: Nieves Atienza, Ken Been, Mike Bekos, Marc Benkert, Kevin and Maike Buchin, Jaroslaw Byrka, Natalia de Castro, Jinhee Chun, Carmen Cortés, M. Ángeles Garrido, Clara I. Grima, Herman Haverkort, Gregorio Hernández, Danny Holten, Michael Kaufmann, Matias Korman, Moritz Kroll, Alberto Márquez, Damian Merrick, Auxiliadora Moreno, Yoshio Okamoto, Sheung-Hung Poon, José Ramon Portillo, Pedro Reyes, Rodrigo Silveira, Antonios Symvonis, Takeshi Tokuyama, Takeaki Uno, Jesús Valenzuela, Maria Trinidad Villar, Markus Völker, and Alexander Wolff. Collaborating and discussing research problems with you has been an enjoyable and rewarding experience, many of the results of which are part of this thesis. Thanks also to my proof readers Robert Görke, Markus Jalsenius, Ignaz Rutter, and Rodrigo Silveira and to the German Research Foundation (DFG) for funding the project GeoNet.

Traveling to many interesting places in the world and meeting new friends and colleagues is one of the pleasures of being a PhD student. In particular, I thank Takeshi Tokuyama for inviting me to spend three weeks in his group at Tohoku University, Sendai, and Takeaki Uno for inviting me to NII, Tokyo. Thanks also to Joachim Gudmundsson for hosting me two weeks at NICTA, Sydney.

Last but not least, I want to thank my family who always believed in me and gave me all the support I needed: my parents, my grandparents, and my brother Robert. A very special thanks goes to my beloved wife Nicole. I don't want to imagine how life would be without you. All the time, but especially during the hard last weeks of writing, you gave me love, patience, encouragement, and the necessary distraction from work. Thank you! Oh, and of course I shall not forget to mention my little son Otto. Although you have been with us for only slightly more than a year by now, you are constantly reminding me of what is really important in life. Holding you in my arms and playing and laughing with you is so simple, and yet makes these moments full of joy and happiness.

Deutsche Zusammenfassung

Visualisieren von Netzwerken ist eine Aufgabe, die in einer Vielzahl von Anwendungsbereichen auftritt, in denen Daten mathematisch als Graph modelliert werden und für einen menschlichen Betrachter veranschaulicht werden sollen. Ziel ist dabei in der Regel, eine möglichst *effektive* Darstellung des Graphen in der Ebene zu finden, also eine Darstellung, die für den Betrachter leicht zu lesen ist und gleichzeitig die zu vermittelnde Information objektiv wiedergibt, sei es zur Exploration von unbekannten Zusammenhängen oder zur Präsentation von bekanntem Wissen.

Die graphische Darstellung eines Knotens bzw. einer Kante des Graphen hängt einerseits von geometrischen Attributen wie Position, Größe oder Form ab und andererseits von rein graphischen Attributen wie Farbe, Muster oder Helligkeit. In meiner Dissertation beschäftige ich mich mit den geometrischen Aspekten der Netzwerkvisualisierung, dem sogenannten *Graphenzeichnen*. In der Regel werden Knoten durch Punkte repräsentiert, deren Koordinaten geeignet zu bestimmen sind. Kanten werden üblicherweise als einfache offene Kurven zwischen ihren Endpunkten dargestellt. Unter anderem unterscheidet man beim Graphenzeichnen zwischen *eingebetteten* Graphen, also Graphen für die bereits eine initiale Zeichnung vorliegt, und abstrakten Graphen, die lediglich durch Knoten- und Kantenmengen definiert sind.

Die Arbeit ist in fünf Themenbereiche gegliedert. Dabei beschäftige ich mich in den ersten drei Bereichen mit Netzwerkvisualisierungsproblemen, die aus unterschiedlichen Anwendungen heraus motiviert sind, während die beiden letzten Themen eher theoretisch motiviert sind.

Schematische Linienpläne. Die erste Anwendung ist das Erstellen von schematischen Karten oder *Linienplänen*, wie sie vorwiegend zur Darstellung von öffentlichen Verkehrsnetzen eingesetzt werden. Hier geht es darum, einen durch die geographische Lage der einzelnen Stationen und ihrer Verbindungen bereits eingebetteten Graphen so zu schematisieren, dass die Lesbarkeit für visuelle Navigationsaufgaben erhöht wird. Dabei werden die zulässigen Kantenrichtungen eingeschränkt, Knicke entlang von Verkehrslinien minimiert und Kantenlängen maßstabsunabhängig vereinheitlicht – ohne allerdings das ursprüngliche Layout zu sehr zu verzerren. Ein besonderes Augenmerk liegt auch auf dem Erstellen von beschrifteten Linienplänen, also Netzwerkvisualisierungen, die ausreichend Platz bieten um alle Stationen überlappungsfrei mit ihrem Namen zu beschriften. Das bereits ohne Beschriftungen als NP-schwer bekannte Problem modelliere ich als gemischt-ganzzahliges lineares Programm und evaluiere den Ansatz anhand von drei Fallstudien. In Bezug auf die Qualität der Ergebnisse ist meine Methode bisherigen Ansätzen weit überlegen.

Ein weiteres Problem, das häufig bei der Visualisierung von Linienplänen auftritt, ist die Darstellung von parallel verlaufenden Verkehrslinien entlang gemeinsamer Kanten. Üblicherweise wird jede Bus- oder Bahnlinie stetig und in einer eindeutigen Farbe entlang der benutzten Kanten des zugrundeliegenden Netzwerks gezeichnet. Dabei kann es – trotz der

Planarität des Netzwerks – zu Kreuzungen kommen. Nur manche dieser Kreuzungen sind durch die Topologie des Netzwerks unvermeidlich. In meiner Dissertation betrachte ich daher erstmals die Minimierung solcher Linienkreuzungen. Mit dynamischer Programmierung wird das Problem optimal für Kreuzungen entlang einer einzelnen Kante im Netzwerk gelöst. In einer verwandten Problemstellung geht man davon aus, dass die Position der Linien in ihren jeweiligen Endstationen bereits gegeben ist. In diesem Fall gebe ich einen Algorithmus zur optimalen Linienanordnung für beliebige planare Netzwerke an.

Dynamische Landkarten. Der zweite Anwendungsbereich entstammt wiederum der Kartographie, diesmal geht es jedoch um die Darstellung dynamischer, interaktiver Karten, in denen der Nutzer kontinuierlich den Kartenausschnitt und -maßstab entsprechend seinen Anforderungen wählen kann. Im Gegensatz zur Visualisierung statischer Karten sind hier viele typische Fragestellungen noch offen. Ich beschäftige mich in dieser Arbeit mit zwei Problemen: der Beschriftung von Punkten und der Generalisierung von Polygonzügen. Unter dem Begriff der *Generalisierung* versteht man in der Kartographie die Anpassung der Darstellungskomplexität von Objekten in der Karte an ihren Maßstab. Das Generalisieren von Polygonzügen (z.B. in Straßen- oder Flussnetzwerken) für einen festen Maßstab ist algorithmisch recht gut verstanden. In Bezug auf das kontinuierliche Zoomen in interaktiven Karten ergibt sich jedoch das Problem, dass es mit den bisherigen Methoden zu Diskontinuitäten kommen kann, z.B. wenn eine Gebirgsstraße beim Herauszoomen für die Darstellung ab einem gewissen Maßstab aufgrund des eingeschränkten Platzes plötzlich eine Serpentine weniger aufweisen soll. Um solche Diskontinuitäten zu vermeiden stelle ich einen Algorithmus vor, der zunächst die beiden Polygonzüge in charakteristische Abschnitte unterteilt und dann eine Zuordnung der Abschnitte bestimmt, die bezüglich eines geeigneten Abstandsmaßes optimal ist. Dadurch wird erreicht, dass bei der Interpolation semantisch äquivalente Abschnitte während des Zoomens ineinander übergehen. Anhand mehrerer Beispiele vergleiche ich die Interpolation entsprechend der berechneten optimalen Zuordnung der Abschnitte mit einer gewöhnlichen linearen Interpolation; dabei zeigt sich, dass die Verfälschung der Polygonzüge bei der vorgestellten Interpolation weit geringer ausfällt als bei einer linearen Interpolation.

Ein zweites Problem im Zusammenhang mit dynamischen Landkarten ist das konsistente Beschriften von Kartenobjekten bei Nutzerinteraktion. Beim Herauszoomen aus einem Kartenausschnitt haben die vorhandenen Ortsnamen bei konstanter Schriftgröße auf dem Bildschirm einen wachsenden Platzbedarf in der Karte, so dass es zu Verdrängungseffekten kommt und bestimmte Namen ausgeblendet werden müssen. Eine Beschriftung heißt *konsistent*, wenn jeder Ortsname in höchstens einem Intervall von Maßstäben sichtbar ist. Die Beschriftungsintervalle sollen so gewählt werden, dass integriert über alle Maßstäbe möglichst viele Ortsnamen sichtbar sind. Ich zeige die NP-Vollständigkeit des Problems und stelle Approximationsalgorithmen für mehrere Varianten vor.

Phylogenetische Bäume. Phylogenetische Bäume sind Binärbäume, die in der Biologie die evolutionären Verwandtschaftsbeziehungen zwischen verschiedenen Spezies darstellen. Ein solcher phylogenetischer Baum ist jedoch meist nur eine Hypothese der realen Abstammungsverhältnisse und so lassen sich für die gleiche Menge von Spezies unterschiedliche Bäume erstellen, die auf ihre Gemeinsamkeiten untersucht werden sollen. Ein visueller Vergleich zweier Bäume ist mit einem *Tanglegram* möglich, einer Gegenüberstellung der beiden planar gezeichneten Binärbäume, so dass die Blätter (die den Spezies entsprechen) auf zwei parallelen Geraden angeordnet sind und je zwei sich entsprechende Blätter miteinander durch eine *Zwischenbaumkante* verbunden werden. Im Allgemeinen kreuzen sich

diese Interbaumkanten. Das Problem ist also, zwei planare Baumlayouts zu finden, die die Kreuzungsanzahl der Interbaumkanten minimieren. Ich zeige die NP-Vollständigkeit bereits für vollständige Binärbäume. Für diesen Fall gebe ich einen Festparameter- und erstmals einen Approximationsalgorithmus an. Außerdem beschreibe ich eine neue Heuristik für allgemeine Binärbäume, die sich in einer experimentellen Evaluation allen bisherigen Algorithmen als deutlich überlegen erweist und oft sogar eine optimale Lösung findet.

Überdeckungskontaktgraphen. Ein klassisches Resultat in der Graphentheorie besagt, dass sich jeder planare Graph als Kontaktgraph von Kreisen in der Ebene darstellen lässt. Andererseits geht es in vielen geometrischen Optimierungsproblemen darum, beispielsweise Punkte durch andere geometrische Objekte (z.B. Kreise oder Polygone) in einer gewissen Weise optimal zu überdecken. Überdeckungskontaktgraphen kombinieren die beiden Fragestellungen: Gegeben ist eine Menge S von zu überdeckenden Objekten (z.B. Punkte) und eine Klasse von Überdeckungselementen (z.B. Kreise). Gesucht ist eine Überdeckung von S, so dass jedes Objekt von genau einem Element überdeckt wird und die Überdeckungselemente sich höchstens berühren. Dadurch wird der Überdeckungskontaktgraph (ÜKG) mit der Knotenmenge S definiert, in dem eine Kante zwischen zwei Knoten genau dann besteht, wenn sich die zugehörigen Überdeckungselemente berühren. Ich betrachte die beiden Fragen, ob zu gegebenem S ein zusammenhängender ÜKG existiert und ob sich ein gegebener Graph als ÜKG auf S realisieren lässt. Ich zeige, dass sich das Zusammenhangsproblem unter bestimmten Voraussetzungen effizient lösen lässt und dass das Realisierungsproblem NP-schwer ist.

Konsistente Strahlen im Raster. Der letzte Teil meiner Arbeit beschäftigt sich mit einem sehr grundlegenden Visualisierungsproblem von gerasterten Strecken in der Gittergeometrie. Übliche Rastermethoden, die auf dem Schnitt von euklidischen Strecken mit einem Pixelgitter beruhen, weisen Konsistenzprobleme auf, wenn mehrere Strecken gleichzeitig betrachtet werden. Beispielsweise ist der Schnitt zweier Strecken nicht immer eine zusammenhängende Pixelmenge. Ich stelle vier Axiome auf, die die Konsistenz von Rasterstrecken garantieren. Für den Spezialfall von Strahlen, die von einem gemeinsamen Ursprung ausgehen, beweise ich eine untere Schranke für den Hausdorffabstand zwischen euklidischem Strahlensegment und zugehöriger konsistenter Rasterstrecke. Gleichzeitig gebe ich ein System von konsistenten Strahlen mit passender oberer Schranke an; das System ist in diesem Sinne also asymptotisch optimal.

Contents

Chapter 1

Introduction

Our world is—and has always been—a world of networks. But only over the last decade *network theory* has become a hot research topic (reflected, for example, in several monographs on network theory and analysis [Bar03, Wat03, BE05, NBW06]), and it has also triggered a lot of media response, especially on social networks: the relationships of the 9/11 terrorists [Kre02], the spread of SARS in the 2002/03 outbreak [MPN+05], the myth of "six degrees of separation" between any two individuals [Wat03] (popular examples link actors through film roles to the American actor Kevin Bacon or mathematicians through co-authorships to Paul Erdős), or even the spread of obesity [CF07] are explained by the analysis of a *network*, that is, a set of *nodes* (in the above examples human individuals) and *links* between them that represent some relationship depending on the context. The growing popularity of social networking web sites like Facebook, MySpace, or LinkedIn additionally increases the public awareness of the networks of our friends, colleagues, and family, which form the invisible basis of our social lives.

But networks span more than "just" our society. One of the first example of a network one might think of is clearly the Internet. Physically, it is a network of computers (the nodes) that are linked by communication channels like broadband cables or satellite links. Additionally, the Internet is often used as a synonymous term for the World Wide Web, which is a network of web sites that are connected by hyperlinks. Social networks also use the Internet infrastructure: one obvious example are the social networking web sites mentioned before, another example is the network based on email contacts in a group of individuals. Similarly to computer networks, there are (mobile) telephone networks, where individual phones are linked to the closest transmission tower or switching center, which in turn is linked to other network nodes. Based on this physical network, especially intelligence services and the police are interested in the network defined by phone contacts between individuals. Further examples are electricity, gas, and water networks, where nodes are, for example, individual households or distribution stations, and links are cables or pipelines. Leaving the area of man-made technological networks, there are networks in biology, for example, protein interaction networks, where nodes are proteins that are linked through the metabolic processes in which they are involved. Evolutionary (or phylogenetic) trees are networks of species connected through common ancestors. In medicine, there is a trend to redefine and group diseases by the collection of genes and proteins that are active in affected cells rather than by the observable symptoms. Just because two tumors look similar under the microscope does not necessarily mean that this holds true on the gene and protein level. This means nothing less than considering diseases as nodes in a network, where one disease is linked to another by a gene that is associated to both of them [GCV+07]. In economics, the relationships between companies, for example, based on

mutual shareholding, form a network just as the trade links do between different countries. In engineering, examples of networks are circuits of electronic components or inheritance relations between classes in a software project. A final example are transportation networks like road networks or public transport networks. Nodes in transportation networks can be road junctions, bus stops, train stations, or airports. In road networks two nodes are connected if there is a road between them, in public transport networks connections are established through a direct transportation service (for example, adjacent stops on a bus or train line or direct flight connections). This list of examples is far from being comprehensive, but it shows how entangled we are in networks of all kinds from the microscopic level of our cells to the macroscopic level of global communication.

One characteristic feature of all these networks is that they are hardly visible to our eyes. Links between two entities are either abstract relations like friendship, hyperlinks, or metabolic processes, or they are concrete links in physical networks like an electricity cable or the asphalt of a road. While abstract relations are inherently invisible, we might be able to see a few links of a physical network. But the full network to which these links belong is usually considered at a regional or even global scale that we cannot overlook—it is simply a matter of size.

How can we still deal with such networks? The answer is *visualization*. Visualization, that is, the graphical representation of the nodes and links of a network, is an essential tool for exploring, analyzing, and communicating networks. A lot of the recent publicity of network theory is not only due to striking scientific results but also to the availability of appealing visualizations that can be used, for example, to illustrate newspaper articles. One should beware, though, of regarding visualization primarily as a means to generate beautiful pictures. It is much more than that. Visualizations aim at faithfully representing the information contained in the given data. A carefully designed visualization is a very efficient means to communicate information, to explore data, and to stimulate new insights—or as Tufte [Tuf01] puts it: "Often the most effective way to describe, explore, and summarize a set of numbers—even a very large set—is to look at pictures of those numbers."

Network visualizations usually depict nodes as dots or disks and links as arcs connecting the corresponding nodes; conversely, this visualization style is known as *node-link diagram*. Furthermore, nodes are usually labeled with the name of the entity that they represent. This is necessary to relate nodes in the visualization to nodes in the actual network. Visual variables like color or size can be used to encode additional node properties. If links are not just binary, but carry a weight or some other information, then the corresponding arcs can also be labeled or varied in their appearance. A visualization example from medical research is given in Figure 1.1, which shows the aforementioned human disease network identified by Goh et al. [GCV+07]. A rather different visualization style, though still a node-link diagram, is used in Figure 1.2, which visualizes protein interactions in molecular pathways of a cancer cell in the style of a schematic metro map [HW02]. These two examples already demonstrate that there is no such thing as a general-purpose visualization style for all networks, not even in the same area of molecular networks in human cells. Good network visualizations are tailored for a specific task and a specific network at hand. While Figure 1.1 is well suited for exploring the network by identifying unknown patterns in the network structure, the main purpose of Figure 1.2 is communicating the authors' findings in a clever way by taking advantage of the readers' familiarity with metro maps.

Why is visualization so important when dealing with networks? What makes a drawing of a network superior to a simple list or matrix of nodes and links? As humans we perceive our environment primarily through vision. Visual perception has evolved over the history

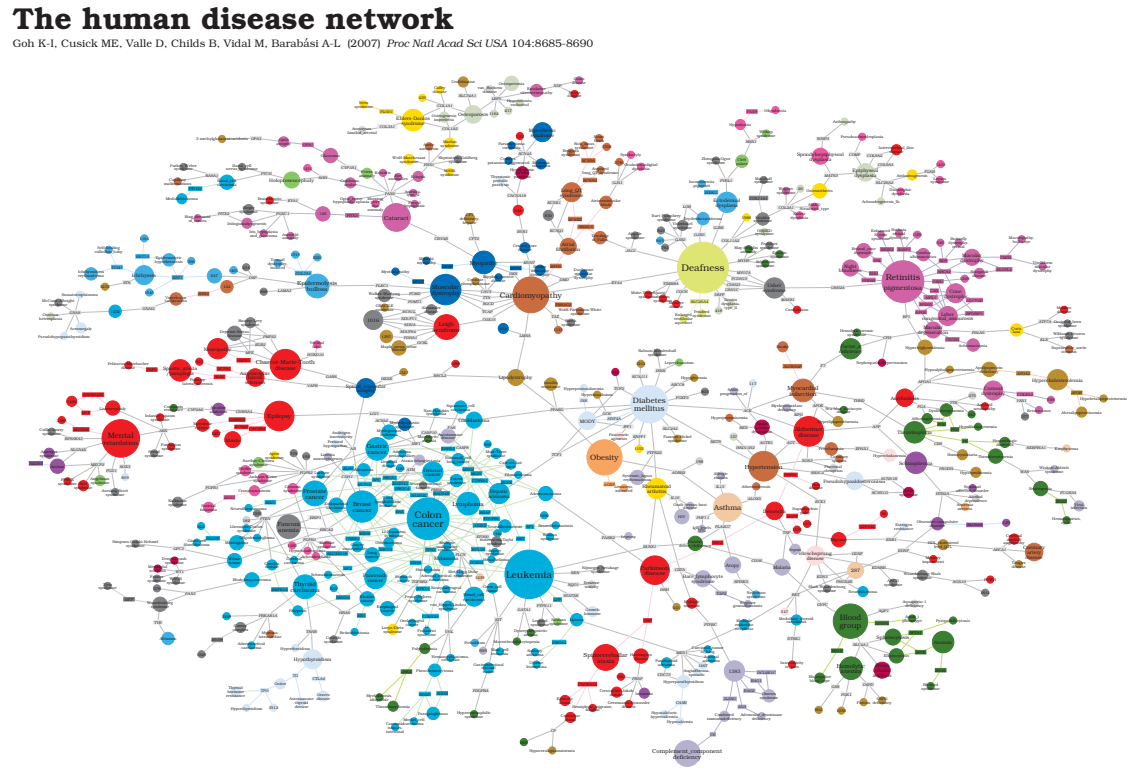


Figure 1.1: Visualization of the human disease network by Goh et al. [GCV+07]. Nodes are shown as disks (diseases) and rectangles (genes). There is a link between a disease and a gene if the gene is associated to the disease. The size of the nodes symbolizes the number of genes that are associated to this disease. Colors indicate a classification of the diseases, for example, light blue identifies cancer.

of mankind into a very powerful system that is highly efficient, for example, in recognizing certain (geometric) patterns in what we see, sometimes even imposing them where they do not exist. The research area of *information visualization* is concerned with mapping abstract data in a meaningful way into a two- or three-dimensional space such that we may quickly identify regularities and irregularities in the data, see the excellent work of Tufte [Tuf01, Tuf90, Tuf97] or the comprehensive collection of classic papers in information visualization by Card et al. [CMS99] for more details. Network visualization is thus a sub-discipline of information visualization. Nodes are placed (mostly) in the plane and are connected by arcs. In a good drawing this allows us to quickly see whether two nodes are connected by a sequence of links, whether a node is linked to a large or a small number of neighbors, whether the local connectivity structure of two nodes is similar, whether there are central nodes that have short distances to most other nodes, which parts are disconnected and so on. Take Figure 1.1 as an example that efficiently provides answers to these questions. At the same time, a bad drawing can inhibit seeing such network properties. For example, spatial proximity of two nodes is often seen as a visual clue for proximity in the network—but in a poor visualization these two concepts of proximity need not correlate.

In a textual or matrix-based description of a network, especially a large network, it is very difficult for a human to infer network properties as those mentioned above. Conversely, computer vision is still bad in discovering unknown patterns in images, whereas many

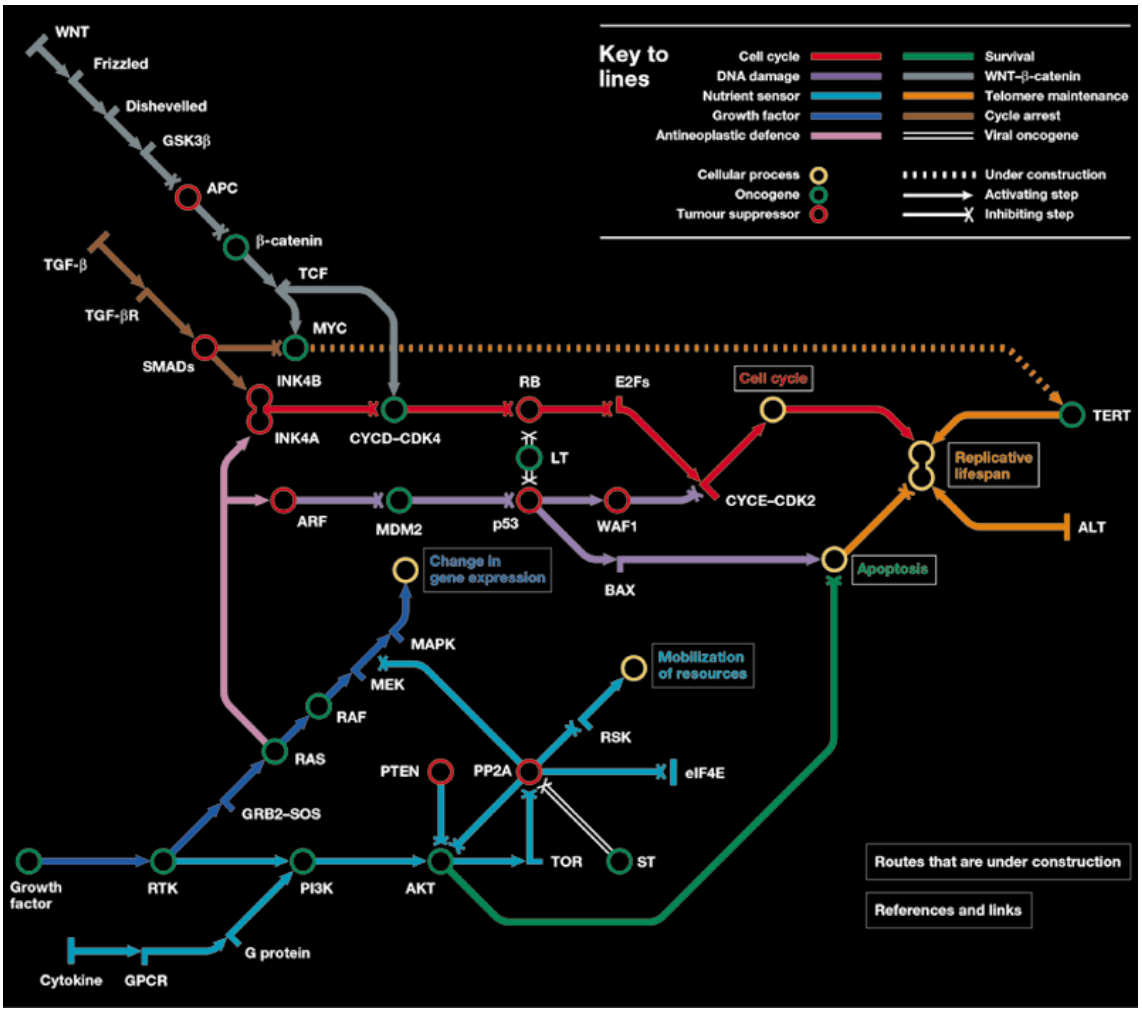


Figure 1.2: Visualization of the protein network of molecular pathways in a cancer cell [HW02]. The metro-map metaphor is used to depict each pathway as a distinctly colored transport line, where two adjacent nodes are related either by inhibition or by activation.

structural network properties like centrality measures or clusterings can be efficiently computed from digital network representations, even for very large networks, see the monograph on network analysis by Brandes and Erlebach [BE05]. This is an ideal precondition for exploiting synergy effects: perform an initial computation of typical attributes in the network and use a suitable algorithm to produce a visualization of the network that highlights these algorithmically obtained features; subsequently, we can apply our visual skills to (interactively) explore the network drawing, discover expected or unexpected patterns, and finally state and confirm a hypothesis about the data.

The use of visualizations for exploring networks predates modern computer visualization techniques, see Freeman's article on the history of visualizing social networks [Fre00]. For example, Moreno [Mor53] describes in his book how he used hand-drawn visualizations to explore relational sociological data in his early studies from the 1930s. In those days working with network visualizations, however, was a time-consuming and tiring trial-and-error process until the drawing was satisfying [BKR+99]. Obviously, manual drawings are feasible only for rather small networks.

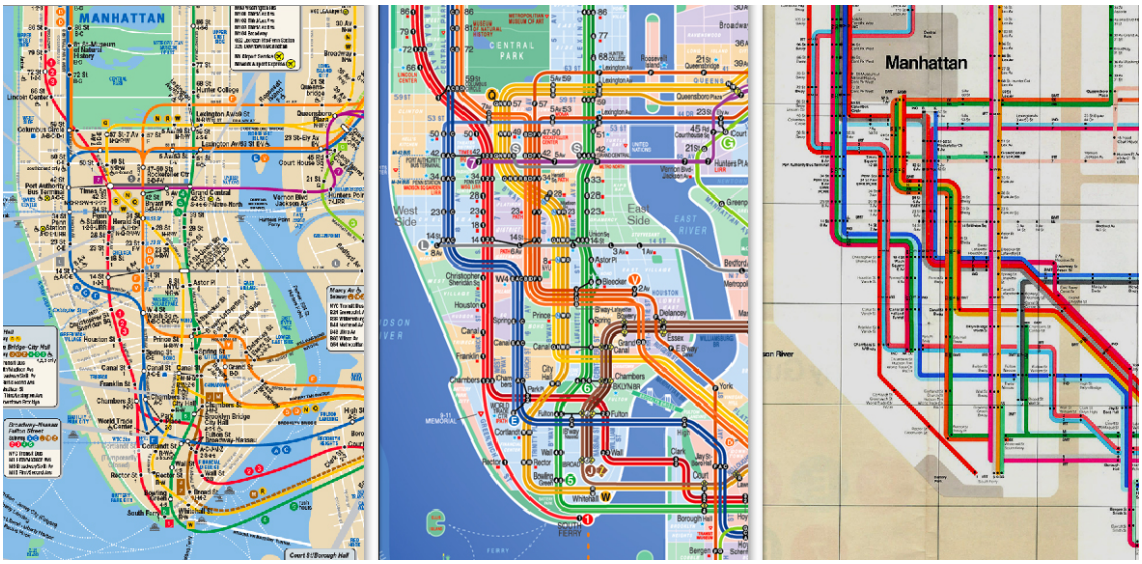


Figure 1.3: Comparison of three subway maps of New York [Kic08]: the current official map with a geographically accurate layout (left), a schematic map proposed by Kick Design (middle), and Vignelli's official schematic map from 1972 (right).

In cartography, visualizations even date back to the earliest map-like wall paintings from the stone age. Maps depict, among other things, various geographic networks, for example, roads, rivers, or region boundaries. The important difference between geographic and abstract networks is that nodes in a geographic network usually represent geographic sites that have a canonical position on a map defined by the map projection. These positions may be altered, for example, in a schematic public transport map, but the acceptable distortion of the original geometry of the network is clearly limited. Figure 1.3 shows a comparison of three different visualizations of the subway network of New York. The map on the left overlays a road map with the geographically accurate course of the subway lines. The map in the middle trades geographic accuracy for clarity of the network; it was recently proposed by the graphic designers of Kick Design [Kic08]. The map on the right is a historic official map by Massimo Vignelli from 1972 that abstracts even more from geography, for example, by simplifying the shape of Manhattan and the East River. All three maps have their advantages and disadvantages depending on the usage of the map. While schematic maps are easier to use for navigating on the subway network, the combined road and subway map is better suited for usage above and below ground.

In the visualization examples given above, we have distinguished between abstract and geographic networks. We can further characterize the examples by the intended usage of the visualization. On the one hand, there are *explorative* visualizations used mainly by researchers and practitioners in order to gain an understanding of the network at hand. In a visual exploration process, the user often invokes certain network analysis algorithms and produces many different visualizations in his search for meaningful patterns. Card et al. [CMS99] use the phrase "*using vision to think*" to describe this visual exploration process. Ideally, the visualizations can be modified interactively, for example, to zoom in at interesting parts or to move some nodes manually. This also reflects Shneiderman's visual information-seeking mantra "*overview first, zoom and filter, then details on demand*" [Shn96]. On the other hand, there are network visualizations with the purpose of *communicating* the network to colleagues, decision makers, or the general public. In

such a case the author or designer of a visualization wants to convey certain aspects of the network by means of a network drawing. A metro (or subway) map, such as the one of New York in Figure 1.3, is a clear example of a visualization intended to *communicate* the network to the passengers. A metro map is not about finding hidden patterns in the network, but about clearly depicting its topology. Above all, it must be an easy-to-use tool for route planning and travel information. Hence, readability and usefulness as a navigational aid should be the prominent aspects in the design of a metro map. But also an initially explorative visualization, such as Figure 1.1, that leads to a researcher's actual insight is suitable as a means of communicating the findings to the readership of an article. If the reader is able to rediscover the same patterns in an illustration that have been discovered by the author, the author's arguments may be much more convincing.

Network visualization is a topic that involves researchers from various fields. Cognitive science can help answering general questions about the perception of visualizations and thus may provide guidelines for designing effective visualizations, that is, visualizations that convey the intended information in an easy-to-read manner. Researchers from the application fields, such as sociology, biology, or cartography, can provide valuable insights into what properties visualizations of networks in their field must highlight and how this should influence the network layout. Computer science, finally, is concerned with mathematically modeling networks as combinatorial structures called *graphs* and, subsequently, with designing algorithms to compute *graph layouts*, that is, positions of nodes and links in the plane (Chapter 2 formally introduces the concepts of graphs and graph layouts). Graph layouts are characterized by their topology and their geometry and we can abstract from many rendering questions that concern only the final display of the network like the use of colors or shapes to represent nodes.

The area of computer science that deals with the theory and algorithmic questions of graph layouts is known as *graph drawing*. The topic is covered by the annual International Symposium on Graph Drawing (GD)[1] and by three monographs [dBETT99, KW01, NR04]. If a network visualization problem is viewed from the perspective of graph drawing, the application domain typically dictates a set of basic constraints for the layout, for example, that links must be drawn as straight lines. The graph layout problem is then formulated as a constrained optimization problem that asks for a graph layout that satisfies the constraints and optimizes one or more aesthetic criteria in order to enhance its readability, for example, minimization of the number of link crossings. We have mentioned node labeling as an important part of network visualizations; labeling adds another level of complexity to the layout problem. For maximum readability the labels must be unambiguously placed at their nodes and they may not occlude other labels or any part of the network itself. Many of the optimization problems arising in the field are NP-hard, that is, there is little hope for efficient exact algorithms. Hence, there is a need for suitable approximation algorithms or good heuristics, especially for large networks. Moreover, network visualizations are nowadays often beyond static graphics printed on paper. In dynamic interactive settings, where users can zoom and pan the display and the visualization is animated, many new problems arise. This applies in particular to the visualization of geographic networks. For example, consider the visualization of an interactive map on a mobile device. Herman et al. [HMM00] surveyed graph drawing from the perspective of information visualization including a discussion on navigation and interaction techniques used in network visualization.

[1] see `http://www.graphdrawing.org`

Thesis Outline

In this dissertation we address five network visualization problems that occur primarily in cartography and biology. We present complexity results for the problems, as well as algorithms to solve them, at least heuristically or approximately. We also address a geometric representation of networks that is different from node-link diagrams, and we investigate problems caused by grid-based display media such as computer screens. The following is a brief summary of the results obtained in the individual chapters.

Chapter 2 – Preliminaries

We introduce the necessary background from information visualization and graph drawing. Furthermore, we repeat the basic concepts of NP-completeness and polynomial-time reductions that are necessary for the understanding of our complexity results. Finally, approximation algorithms, fixed-parameter algorithms, and mathematical programming are introduced as important concepts to deal with NP-hard problems.

Chapter 3 – Metro Maps: Layout and Labeling

We investigate the layout and labeling of schematic public transport maps, also known as *metro maps*. The input is a geographic network whose nodes and links need to be placed with as little distortion as necessary such that all lines in the layout are horizontal, vertical, or 45°-diagonal and typical aesthetic criteria of metro maps are optimized. We describe a set of design rules obtained from a careful inspection of real-world maps and subsequently model the layout problem as a mixed-integer program. We also show how the space requirements for station labels can be taken into account. In a detailed case study we evaluate the applicability of our method on the basis of three real-world example networks by comparing our results to the official metro maps and, where available, to the results of previous algorithms. In terms of visual quality and conformance to the design rules our method is superior to previous approaches.

Chapter 4 – Metro Maps: Line Crossings

Our second problem arises again in the visualization of metro maps. Public transport networks frequently have the property that parts of the network infrastructure are used by multiple transport lines. In the visualizations this fact is usually displayed by bundles of colored parallel lines (each of which represents one transport line) drawn along the underlying network links. Two lines in a bundle cross if their relative order is changed between two stations. We were the first to consider the problem of minimizing these line crossings, and we give exact algorithms for two variants of the problem.

Chapter 5 – Dynamic Maps: Morphing Polylines

Next, we study a problem that arises in interactive dynamic maps where users can zoom continuously. In such maps the level of detail of the displayed content must be adapted continuously to the selected scale. This process is known as *generalization* in cartography. A special case is the generalization of polygonal chains (also called *polylines*). For static maps there are established line simplification algorithms for that purpose, but they are not suitable to produce smooth animations for continuous zooming. We present a dynamic-programming algorithm that computes an interpolation between two representations of the same polyline but at different scales. Our interpolation minimizes the visible movement

and maps semantically equivalent parts of the polylines to each other. Since links in a road or river network are typically represented as polylines, we can apply our algorithm to each individual network link; this extends our interpolation to whole geographic networks. The successful applicability of our method is finally exemplified in a case study for real-world road, river, and region-boundary data; our method produces interpolations with far less distortion than regular linear interpolations.

Chapter 6 – Dynamic Maps: Labeling

Another problem arising in dynamic maps is the labeling of network nodes or map features in general. In static maps, the basic requirements for labeling are that features are unambiguously labeled and that no two labels overlap. Dynamic maps add new constraints to label placement, namely that labels do not suddenly change their positions and that labels do not flicker, that is, disappear and reappear several times during zooming and panning. Under these constraints the goal is to maximize the number of visible labels—analogously to static map labeling. We model the dynamic label placement problem in three dimensions, where scale is the third dimension. We prove the hardness of this new problem even for quite simple variants and give a collection of approximation algorithms.

Chapter 7 – Optimal Tanglegram Layout

Tanglegrams are pairs of binary trees on the same set of leaves, as they appear, for example, as phylogenetic trees in biology. Such trees represent different evolutionary hypotheses. Biologists need to find similarities and differences of the trees in order to assess their plausibility. In a tanglegram drawing, the two trees face each other with their leaves arranged on two parallel lines. Each leaf of one tree is connected to the corresponding leaf in the other tree by an *inter-tree edge*. We study the problem of minimizing the number of inter-tree edge crossings, show its NP-hardness even for complete binary trees, and give exact, approximate, and heuristic algorithms. We compare our algorithms in a first comprehensive experimental study with each other and with previous algorithms. Our heuristic is far superior to its competitors and even finds optimal solutions in many cases.

Chapter 8 – Cover Contact Graphs

Here, we deal with an alternative representation of networks, where nodes are represented as geometric objects (for example, disks) and a link between two nodes is realized if two objects touch each other; such a representation is called a *contact graph*. Additionally, we are given a set of geometric seed objects in the plane (for example, points) and require that each node object *covers* (that is, contains) exactly one seed object. In this case, the set of objects form a so-called *cover contact graph* (CCG). We are interested in two questions for this new class of geometric graph representations: (a) for a given set of seeds, is there a connected CCG and (b) can a given network be realized as a CCG on a given set of seeds? We show that under certain conditions, we can find a connected CCG efficiently and we show that the realization problem is NP-hard.

Chapter 9 – Consistent Digital Rays

Finally, we consider a very basic and classic visualization problem, namely how to represent line segments (in particular, ray segments) on a grid-based medium like a computer screen; these representations are called *digital* line segments. Conventional methods generate

visually pleasing representations, but these representations often have consistency problems if multiple line segments interact; for example, the intersection of two digital line segments might be a disconnected set of grid points. We propose a set of simple axioms that consistent line segments must satisfy. We show a lower bound on the Hausdorff distance between Euclidean and corresponding consistent digital line segments for the special case of rays from a fixed origin. At the same time, we give a construction of a family of digital rays that realizes this lower bound asymptotically, that is, our digital rays are asymptotically worst-case optimal.

Chapter 2

Preliminaries

In this chapter we give a brief overview over the main concepts in visualization, graph theory and graph drawing, complexity theory, and algorithms for NP-hard problems that are referred to in this work. We do, however, assume a basic knowledge of algorithmic and complexity-theoretic concepts (for example, the big-O notation for worst-case running times). Introductions to these concepts are found in the standard textbooks on algorithms and complexity theory, see, for example, Cormen et al. [CLRS01] and Garey and Johnson [GJ79].

2.1 Visual Variables

Visualization and graphic design are broad research fields in their own right, and it is clearly beyond the scope of this thesis to review the state of the art in information visualization. Entry points into the literature are Tufte's classic books on information design [Tuf01, Tuf90, Tuf97], the collection of articles by Card et al. [CMS99], a survey chapter by Görg et al. [GPQX07], and Ware's textbook [War04], which considers the topic from the perspective of human perception.

We restrict ourselves in this short section to the introduction of Bertin's concept of *visual variables*, which is very useful to separate the network visualization problem into an algorithmically interesting layout problem, and an independent rendering problem. Jacques Bertin, a French cartographer, published one of the first theoretic foundations of information visualization based on his cartographic experience in his seminal book "Sémiologie graphique" (1967), which was later translated into German and English [Ber83]. He presented a coherent framework of how to encode information in the symbols used in maps or other graphic representations. The position of a symbol is defined by its x- and y-coordinates (in three-dimensional graphics by x-, y-, and z-coordinates). This corresponds to two (or three) *positional variables*. In addition, Bertin defined the six *retinal variables size*, *value* (brightness), *texture*, *color*, *orientation*, and *shape*. Figure 2.1 illustrates how these variables affect the appearance of a symbol. About 30 years later, MacEachren [Mac95] extended the original set of variables, which Bertin intended for paper maps, in order to reflect the potential of new electronic display media. Examples of the extended set of visual variables are *transparency* and *crispness*, as well as acoustic and temporal variables that must be observed over time and require more user attention. In this thesis we restrict ourselves to visualizations that use the original set of visual variables and hence can (in principle) be printed on paper.

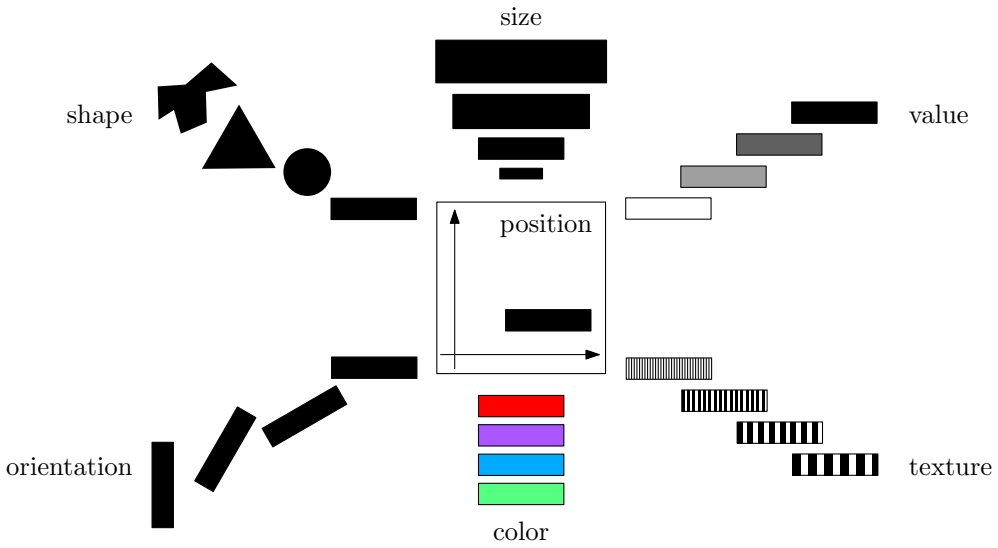


Figure 2.1: Bertin's visual variables divided into positional variables (center) and retinal variables [Ber83].

In terms of network visualization, the retinal variables can be used to represent additional attributes of nodes and edges in a drawing. Numeric attributes, such as node and edge weights, are best visualized using the variables *size* and *value* that are perceived in an ordered fashion; the variable *size* can even encode quantitative information, that is, we can infer numeric differences from size differences. For categorical attributes, such as group memberships of nodes, the four remaining retinal variables should be used.

The positional variables are used to actually place the nodes and links in the plane. The topology and the geometry of a network visualization are fully determined by the positional variables as long as the retinal variables *shape*, *size*, and *orientation* remain unused. This is the case, for example, in the popular *node-link diagram* visualizations, where nodes are represented as points or unit-size disks and links are represented as arcs.

Hence, for node-link diagrams, we can divide the network visualization problem into a *layout problem* that is concerned with the positioning of nodes and links, and an independent *rendering problem* that is concerned with assigning color, value, and texture to the graphical symbols. In this thesis we are focused on algorithmic solutions for the layout problem that determine the geometric and combinatorial structure of the network visualization. Solutions to the rendering problem, for example choosing a set of colors, can be obtained by applying the general guidelines for graphic design [Tuf90]. Note that, although the majority of research in graph drawing deals with the layout problem, there are also a few algorithmic approaches to the rendering problem, for example, Dillencourt et al. [DEG07] compute node colors so that perceptually dissimilar colors are assigned to connected nodes.

2.2 Graphs and Graph Drawing

We have already mentioned in Chapter 1 that networks are mathematically modeled as graphs and that graph drawing is the research area in computer science that deals with computing and analyzing layouts of graphs. In this section we introduce some basic concepts from graph theory and graph drawing.

2.2.1 Graphs

Although the notions *graph* and *network* are often used synonymously, we refer to a network as the concrete relational node and link data in an application, for example, a road network or a social network. A *graph*, on the other hand, is an abstract mathematical structure that consists of two sets: a set of *vertices* that correspond to the nodes of a network, and a set of *edges* that correspond to the links. We denote a graph as a tuple $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of vertices and $E = \{e_1, e_2, \ldots, e_m\}$ is the set of edges. Each edge $e \in E$ is an unordered pair $e = \{u, v\} \subseteq V$ of distinct vertices $u \neq v$. If the graph is *directed*, that is, the relation between vertices is not symmetric, then we write $e = (u, v)$ for an edge directed from u to v. In both cases, we often use the abbreviation uv to denote the undirected or directed edge between u and v; for an undirected graph we have $uv = vu$. If not stated otherwise, all our graphs are undirected. We may assign vertex weights to a graph by defining a function $\omega : V \to \mathbb{R}$ that assigns to each vertex v the weight $\omega(v)$. Similarly, we may assign edge weights to a graph by defining a function $\hat{\omega} : E \to \mathbb{R}$ that assigns to each edge e the weight $\hat{\omega}(e)$. Two vertices u and v are called *adjacent* if there is an edge $uv \in E$. The edge uv is *incident* to its end vertices (or *endpoints*) u and v, as well as to all other edges incident to u or v. The *degree* $\deg(v)$ of a vertex v is the number of edges that are incident to v. The degree of G is the maximum degree of its vertices. A graph is *vertex-labeled* if there is a function $\lambda : V \to \Lambda$ that assigns a label $\lambda(v)$ to each vertex v, where $\lambda(v)$ is an element in the set Λ of labels. Labels are typically strings or integers. Analogously one could also assign labels to edges.

A *path* in G is a sequence $P = (v_0, v_1, \ldots, v_k)$ of vertices such that $v_i v_{i+1} \in E$ for $0 \leq i \leq k-1$, that is, there is an edge between any two consecutive vertices in P. The *length* of P equal k, the number of edges in P, and it is denoted as $|P| = k$. A path P *contains* a vertex v (written $v \in P$) if $v = v_i$ for some $0 \leq i \leq k$; P *contains* an edge e (written $e \in P$) if $e = v_i v_{i+1}$ for some $0 \leq i \leq k-1$. If all vertices of P are distinct, we say that P is a *simple* path. A *subpath* P' of P is a contiguous subsequence of the vertices of P, that is, $P' = (v_i, v_{i+1}, \ldots, v_j)$ for some $0 \leq i \leq j \leq k$. If $v_0 = v_k$ then P is called a *cycle*; a cycle is *simple* if the subpath $(v_1, \ldots, v_k)$ is simple. A graph that does not contain any cycles is called *acyclic*.

We say that a graph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. For a subset $V' \subseteq V$ of the vertices of G the subgraph *induced* by V' is the graph $G[V'] = (V', \{uv \in E \mid u, v \in V'\})$. A graph G is *connected* if there is a path between any pair of vertices in G. Otherwise the graph is *disconnected* and consists of two or more *connected components*, that is, inclusion-maximal connected subgraphs of G.

There are some subclasses of graphs that are relevant in this thesis. A graph $G = (V, E)$ is called *bipartite* if the vertex set V can be partitioned into two disjoint sets A and B, $A \cup B = V$, such that for every edge $uv \in E$ we have $u \in A$ and $v \in B$ (or vice versa), that is, no two vertices in A and no two vertices in B are adjacent.

An acyclic graph is called a *forest*, and a connected forest is called a *tree*. Tree vertices v with $\deg(v) = 1$ are called *leaves*, vertices v with $\deg(v) > 1$ are called *internal* vertices. A tree $T = (V, E)$ is *binary* (or, in general, d-ary for $d \geq 2$) if $\deg(v) \leq 3$ ($\deg(v) \leq d + 1$) for every vertex $v \in V$. A *rooted* d-ary tree $T = (V, E)$ is a d-ary tree with a distinguished root vertex $r \in V$ of degree $\deg(r) \leq d$. Vertices in rooted trees are commonly called *nodes*[1] [CLRS01] and we adopt this notation. The *depth* of a node v in a rooted tree T is the length of the unique shortest path from v to the root r, and the *height* of T is equal

[1] The notion *node* is thus used for both network nodes and vertices in rooted trees. The correct meaning will be clear from the context.

to the maximum depth of its nodes. A tree T is called *complete* if the depth of all leaves equals the height of T. A node $u \neq v$ on the path from a node v to the root is called an *ancestor* of v, and, conversely, v is called a *descendant* of u. A *direct* ancestor (descendant) of a node v is an ancestor (descendant) u such that $uv \in E$. The direct ancestor of a node v is called its *parent*, the direct descendants of v are called its *children*. Two nodes with the same parent are *siblings*. For two nodes u and v the *lowest common ancestor* $\text{lca}(u, v)$ of u and v is the unique node with maximum depth that is an ancestor of both u and v. Finally, the *subtree rooted at* v is the subgraph of T that is induced by v and the descendants of v.

2.2.2 Graph Drawing

Two-dimensional graph drawing is concerned with embedding a graph $G = (V, E)$ in the plane $\mathbb{R}^2$. In the popular *node-link diagram* representation style the *drawing* (or *layout*) of a graph G is a function $\Gamma : V \cup E \to \mathbb{R}^2$ that maps each vertex $v \in V$ to a point $\Gamma(v) \in \mathbb{R}^2$ and each edge $uv \in E$ to a simple open curve $\Gamma(uv)$ with endpoints $\Gamma(u)$ and $\Gamma(v)$. If an edge uv is drawn as a straight-line segment we use the notation $\Gamma(uv) = \overline{\Gamma(u)\Gamma(v)}$ to denote the straight-line segment between $\Gamma(u)$ and $\Gamma(v)$. In terms of the visual variables introduced in Section 2.1, a drawing Γ thus assigns the positional variables in the underlying network visualization problem. Although a graph and its layout are two different objects, we often do not distinguish explicitly between the two. For instance, we may say "the edge e is a straight-line segment" meaning "the graphical representation $\Gamma(e)$ of the edge e is a straight-line segment." It will be clear from the context whether we refer to the graph or to its drawing.

A drawing Γ is called *planar* if for any two edges e and e' in E the curves $\Gamma(e)$ and $\Gamma(e')$ do not intersect. A graph that admits a planar drawing is called a *planar* graph. A planar drawing Γ subdivides the plane into topologically connected regions called the *faces* of Γ. There is one unbounded region called the *external* face; all other faces are called *internal* faces. Two faces are *adjacent* if their boundaries intersect in an edge. Edges and vertices on the boundary of a face are *incident* to that face.

A planar drawing Γ induces for each vertex v a counterclockwise circular ordering of its incident edges. We can define an equivalence relation on the set of planar drawings of a planar graph G, where two drawings are considered equivalent if for all vertices of G the circular orderings of their incident edges are identical. Each equivalence class of this relation is called an *embedding* and contains all topologically equivalent drawings of G. A planar graph together with an embedding is called an *embedded* planar graph or simply a *plane* graph.

If in a drawing Γ of a graph G all edges are drawn as straight-line segments between their endpoints, then this graph and its drawing are called a *geometric graph*. If such a drawing Γ is planar, G and Γ are called a *plane geometric graph*. Note that in a geometric graph the vertex positions determine the full graph layout.

The general *graph layout problem* is easy to state:

> Given a graph G, find a *nice* drawing Γ of G.

The whole difficulty of the problem lies in the word "nice". We need to map the nodes and links of a network, which usually stems from some application domain, into the plane such that the resulting visualization highlights the desired features of the network at hand (or stimulates the identification of hidden features) and additionally satisfies the domain-specific aesthetic requirements. As Eppstein put it: "If it's important that a graph has

certain properties, it's important to communicate those properties in a drawing" [Epp08]. Now the graph properties and layout requirements can be very different depending on the network itself and the specifics of the application domain. For example, layouts of directed acyclic graphs are often drawn hierarchically with all edges pointing into the same direction in order to show the graphs' acyclicity; planar graphs are usually required to be drawn without edge crossings. Also the domain-specific aesthetics can differ. For example, in some domains compact drawings are important, while others require that edges are drawn as straight-line segments or rectilinearly. It is clear that, for example, the layout of a phylogenetic tree (as in Chapter 7) differs considerably from the layout of a metro map (as in Chapter 3). Tree layouts are usually expected to be layered hierarchical drawings with the root as the topmost node. In a metro map, on the other hand, preserving a general sense of a city's geography is required so that users who know the city can quickly locate their departure and arrival stations on the map. Hence it is important in order to make good drawings that "you understand what [application-specific] structure in the graph you want to convey and what kind of drawing will best convey it" [Epp08].

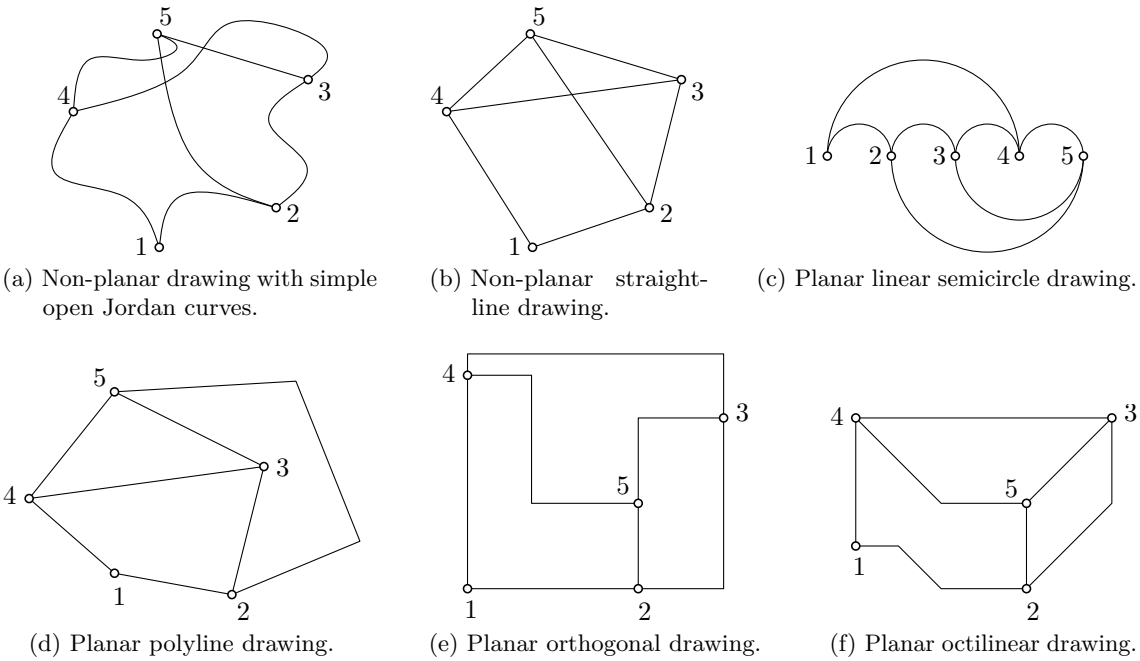
Di Battista et al. [dBETT99, Chapter 2] distinguish *drawing conventions*, *aesthetics*, and *constraints* as three classes of requirements for a "nice" drawing. *Drawing conventions* are fundamental rules that must be globally satisfied by the drawing in order to be admissible. Common examples are orthogonal drawings, octilinear drawings (edges are orthogonal or 45°-diagonal), straight-line drawings, polyline drawings, grid drawings, or planar drawings. Figure 2.2 shows six drawings of a planar graph according to different drawing conventions. For example, straight-line, orthogonal, and octilinear drawings all belong to the more general class of polyline drawings, in which each edge is drawn as a sequence of line segments joined by so-called *bends*. In Figure 2.2c, for example, the vertices are linearly arranged on a line and edges are drawn as semicircles above or below that line. *Aesthetics* specify optimization criteria whose degree of compliance affects the readability of a drawing under some drawing convention. Typically these comprise minimizing edge crossings in a non-planar drawing, minimizing area or total edge length, minimizing the variance of edge lengths, minimizing the number of edge bends, maximizing the angular resolution, or maximizing symmetries in the drawing [BFN85, STT81]. Purchase et al. [PCJ96] performed an empirical study that showed the positive effects of minimizing crossings and bends on the readability of graph drawings. Finally, *layout constraints* affect local properties of the drawing, for example, they can restrict the relative positions of a subset of vertices or edges. In terms of these three concepts, the graph layout problem is reformulated as follows:

> Given a graph G, find a *nice* drawing Γ of G under the given drawing conventions that satisfies all layout constraints and optimizes the aesthetics.

Many of the aesthetic criteria lead to NP-hard optimization problems, at least for general graph classes or in the variable-embedding scenario. Additionally, not just one but several aesthetic criteria are usually relevant for the application. Different aesthetics are often in conflict with each other; hence a weighted sum of criteria or a prioritized sequential optimization can be used to compute a well-balanced and "nice" layout.

We mention two popular algorithmic frameworks for graph drawing that are occasionally referred to in this thesis. A more comprehensive overview of algorithms is found in the standard textbooks on graph drawing [dBETT99, KW01, NR04].

A popular method that is applicable to all kinds of graphs is the *spring-embedder* (or *force-directed*) method. It is based on a physical analogy: vertices are considered as objects that repel each other like electrically charged particles, and edges are considered as springs



(a) Non-planar drawing with simple open Jordan curves.

(b) Non-planar straight-line drawing.

(c) Planar linear semicircle drawing.

(d) Planar polyline drawing.

(e) Planar orthogonal drawing.

(f) Planar octilinear drawing.

Figure 2.2: Six drawings of a planar graph. Note that the embeddings in (c) and (d) are the same, and that the embeddings in (e) and (f) are the same.

that connect two objects and hence impose an attracting force on them. Using iterative optimization methods like gradient descent or simulated annealing, an equilibrium of this physical system is computed. As an example of a typical layout produced by a spring embedder see Figure 1.1 in Chapter 1. Additional optimization criteria can easily be implemented in the existing framework as long as they are modeled as physical forces. Spring embedders are attractive since they do not impose any restrictions (for example, planarity or acyclicity) on the graphs—any graph can be drawn. The aesthetic criteria are modeled implicitly through the forces in the physical system. These forces can be weighted according to the respective importance of each criterion. Spring-embedder visualizations are implemented in most of the available graph visualization tools [Wil01].

The topology-shape-metrics approach [Tam87] is a popular framework for computing orthogonal (or *rectilinear*) layouts. It proceeds in three steps. In the first step the graph is planarized by computing an embedding that uses a small number of edge crossings; each edge crossing becomes a dummy vertex in an equivalent augmented planar graph. The rectilinear crossing minimization problem is NP-hard [GT01]. The second step determines an orthogonal representation that defines the sequence of bends along each edge such that the total number of bends is minimum. The bend minimization problem can be solved efficiently using a network flow algorithm [Tam87, GT96]. Finally, in the compaction phase, the coordinates of vertices and bends are determined such that the resulting drawing has minimum possible area; dummy vertices are finally removed. The compaction problem can again be solved by a network flow algorithm [Tam87, GT96]. In the topology-shape-metrics framework a prioritization of three aesthetics is used: crossing minimization is the most important criterion; for the crossing-minimal embedding a representation with minimum number of bends is computed; finally, this representation is drawn in the minimum possible area.

In the discussion so far, we assumed a node-link diagram representation of the graph, in which the layout problem and the rendering problem are independent. We note, however, that there are also graph representations in which vertices have different sizes or vertices and edges are labeled. Then, ideally, the known space requirements of vertex symbols or labels are already taken into account when solving the layout problem. For placing labels, this extended layout problem is called *graph labeling* [KM99a]. If, in contrast, the layout is computed independently of the prospective space consumption, it may happen that not all labels can be placed or that vertex symbols overlap.

Other forms of graph representations comprise contact or intersection graphs, where vertices are represented as geometric objects (symbols) and edges are realized as contacts or intersections of two such symbols. Contact representation of graphs are studied in Chapter 8. For rooted trees there is the *Treemap* representation [Shn92] in which tree nodes are represented as (nested) rectangles and edges as inclusions between rectangles, that is, each parent rectangle contains all child rectangles (and consequently all its descendants).

2.3 Complexity

Many of the problems considered in this thesis turn out to be NP-hard or NP-complete. The theory of NP-completeness is a basic part of theoretical computer science and introductions are found in various textbooks, for example, in the classic books by Garey and Johnson [GJ79] or Cormen et al. [CLRS01, Chapter 34]. Here, our aim is to provide an intuition of the complexity classes $\mathcal{P}$ and $\mathcal{NP}$, as well as of the important concept of a polynomial-time reduction. For a detailed and formal introduction we refer to the textbooks mentioned before.

The class $\mathcal{P}$ consists of all computational problems that are solvable in polynomial time by a deterministic algorithm, that is, their time complexity is $O(n^k)$, where n is the input size and k is some constant independent of n. An algorithm in the class $\mathcal{P}$ is also called an *efficient* algorithm, although this is to be understood in a theoretical sense only. Algorithms with a time complexity of, say, $\Theta(n^{100})$ are not at all efficient in practice. The class $\mathcal{NP}$, on the other hand, consists of the problems that are non-deterministically solvable in polynomial time. This definition is equivalent to the somewhat more tangible definition that a problem is in $\mathcal{NP}$ if any potential solution can be *verified* in polynomial time and if there is a positive probability of guessing a correct solution (in case there is one). Clearly, any problem in $\mathcal{P}$ is also contained in $\mathcal{NP}$, so we have $\mathcal{P} \subseteq \mathcal{NP}$. One of the most important and famous open questions in theoretical computer science is whether the converse holds or not, that is, whether $\mathcal{P} = \mathcal{NP}$ or not. It is widely believed that $\mathcal{P} \neq \mathcal{NP}$.

As an example, consider the well-known Boolean satisfiability problem 3-SAT. In an instance of 3-SAT we are given a Boolean formula φ in conjunctive normal form (CNF) with (at most) three literals per clause, that is, $\varphi = c_1 \wedge c_2 \wedge \cdots \wedge c_m$ is the conjunction of the set of clauses $C = \{c_1, c_2, \ldots, c_m\}$. Each clause $c_i \in C$ is a set of three literals $c_i = \{l_{i,1}, l_{i,2}, l_{i,3}\}$ corresponding to the disjunction $(l_{i,1} \vee l_{i,2} \vee l_{i,3})$. Each literal $l_{i,j}$, finally, is a variable or the negation of a variable in the set $U = \{x_1, x_2, \ldots, x_n\}$ of variables. We call φ a 3-CNF formula. The formula φ is *satisfiable* if there is a truth value assignment for the variables in U such that φ evaluates to *true*. Obviously, 3-SAT is a problem in the class $\mathcal{NP}$ since it is straight-forward to verify whether a given variable assignment satisfies all clauses and since guessing a particular variable assignment has probability $1/2^n$.

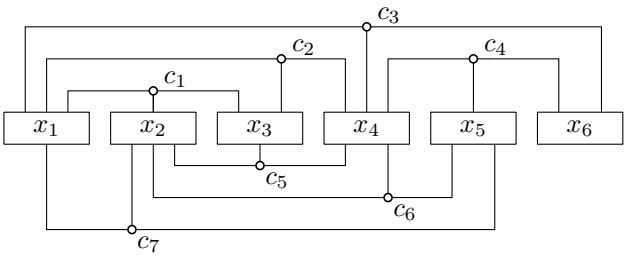


Figure 2.3: Variable-clause graph of a planar 3-SAT formula.

Next, we introduce the notion of *reducibility* among problems, which in turn leads to the fundamental concepts of NP-hardness and NP-completeness. In the theory of NP-completeness, we usually deal with decision problems[2], that is, problems where the answer to any input is "yes" or "no". Very briefly, this allows to relate the problem to the theory of formal languages by encoding input instances as words over some alphabet Σ. The "yes"-instances define a language $L \subseteq \Sigma^\star$ and the decision problem reduces to deciding whether the word corresponding to a given input instance is contained in L.

So consider a decision problem Q. We say that Q is *polynomial-time reducible* to another decision problem R, in short $Q \leq_P R$, if there is a polynomial-time algorithm $\mathcal{A}$ that transforms any input instance I of Q into an input instance $\mathcal{A}(I)$ of R such that $\mathcal{A}(I)$ is a "yes"-instance of R if and only if I is a "yes"-instance of Q. In other words, this means that we can solve problem Q for an instance I by first applying algorithm $\mathcal{A}$ to I and subsequently solving problem R for $\mathcal{A}(I)$.

Now a problem Q is said to be *NP-hard* if any problem in $\mathcal{NP}$ is polynomial-time reducible to Q, that is, Q is at least as hard as any problem in $\mathcal{NP}$. If additionally $Q \in \mathcal{NP}$ we say that Q is *NP-complete*. The most important implication of the notion of NP-completeness is that finding a polynomial-time algorithm for a single NP-complete problem immediately implies $\mathcal{P} = \mathcal{NP}$. Moreover, to show the NP-hardness of a problem Q it suffices to reduce a single known NP-hard problem R to Q, that is, to show $R \leq_P Q$. The fact that any problem in $\mathcal{NP}$ can be reduced to Q follows from the transitivity of the reducibility relation $\leq_P$.

The 3-SAT problem mentioned before is a well-known NP-complete problems [GJ79]. We close this section by presenting the closely related satisfiability problem PLANAR3-SAT, which is used in most of the NP-hardness proofs in this thesis. An instance of PLANAR3-SAT is a Boolean 3-CNF formula φ as in the 3-SAT problem that has the additional property of being *planar*. The planarity of φ is defined through the planarity of the induced *variable-clause graph* $G_\varphi = (V_\varphi, E_\varphi)$, where the set of vertices $V_\varphi = U \cup C$ is the set of variables and clauses of φ and the set of edges $E_\varphi = \{xc \mid x \in U, c \in C, \{x, \neg x\} \cap c \neq \emptyset\} \cup \{x_i x_{i+1} \mid 1 \leq i \leq n-1\} \cup \{x_n x_1\}$ connects each clause to the three variables that appear in it; additionally, E_φ connects all variables in a cyclic manner. Lichtenstein [Lic82] showed the NP-completeness of PLANAR3-SAT. Knuth and Raghunathan [KR92] showed that the variable-clause graph G_φ of a planar 3-CNF formula φ can be drawn on a grid of polynomial size as follows: the vertices of the variables are boxes arranged on a horizontal line and the vertices of the clauses are drawn as the centers of properly nested three-legged combs that attach to the respective variables either from above or from below, see the

[2]Note that an optimization problem Q can be transformed into a corresponding decision problem Q' by asking whether there exists a solution of Q for which the value of the objective function is above (or below) some threshold value K.

example in Figure 2.3. The edges that cyclically connect the variables are omitted, but it is clear that they can be inserted into the drawing without causing any edge crossings. The geometric nature of such a drawing of the variable-clause graph can be exploited in many NP-hardness proofs that involve some geometric reasoning, see Chapters 6 and 8.

The usual approach for reductions from Planar3-Sat is to model each variable vertex as a geometric *variable gadget* that can geometrically encode the values *true* and *false*. The legs of the combs are modeled as *literal gadgets* that absorb the truth values of the respective Boolean literals and transmit them into the *clause gadgets* that model the clause vertices of the variable-clause graph. Now the trick is to design the geometric gadgets such that the resulting overall geometric structure has a certain property if and only if the underlying Boolean formula is satisfiable. This means that the geometric decision problem for this property is NP-hard.

2.4 Approaches for NP-hard Problems

In the final section of this chapter we mention three algorithmic concepts to deal with NP-hard problems. First of all, if exact solutions cannot be computed efficiently then at least computing provably good suboptimal solutions might be feasible. Second, it might be possible to efficiently find exact solutions for some relevant subset of input instances. Third, the size of relevant input data might be small enough to use a general (but in the worst case exponential-time) solution method for the problem. For further reading on algorithmic approaches for NP-hard problems we refer to Hromkovic [Hro03].

2.4.1 Approximation Algorithms

The fact that a problem is NP-hard does not necessarily mean that it is hopeless to tackle it algorithmically. Approximation algorithms are one way to face NP-hard problems by computing near-optimal solutions efficiently. So let Q be an NP-hard optimization problem, that is, an optimization problem whose corresponding decision problem is NP-hard, and let I be an input instance for Q. We assume that we have a polynomial-time algorithm $\mathcal{A}$ that computes feasible, but in general non-optimal solutions for Q. By $\mathcal{A}(I)$ we denote the value of the objective function for the solution that algorithm $\mathcal{A}$ computes for input I. The value of the objective function for the optimal solution of instance I is denoted as $\mathrm{OPT}(I)$.

Let's first consider a minimization problem. This means that $\mathrm{OPT}(I) \leq \mathcal{A}(I)$ for all input instances I. Algorithm $\mathcal{A}$ is called a *factor-ρ approximation algorithm* (or simply a *ρ-approximation*) for $\rho \geq 1$ if

$$\mathrm{OPT}(I) \leq \mathcal{A}(I) \leq \rho \cdot \mathrm{OPT}(I)$$

for any input I. For a maximization problem, we have conversely $\mathcal{A}(I) \leq \mathrm{OPT}(I)$ for all I. Algorithm $\mathcal{A}$ is called a *factor-$(1/\rho)$ approximation algorithm* (or *$(1/\rho)$-approximation*) for $\rho \geq 1$ if

$$(1/\rho) \cdot \mathrm{OPT}(I) \leq \mathcal{A}(I) \leq \mathrm{OPT}(I)$$

for any input I. Clearly, it is desirable to have ρ as small as possible in order to obtain solutions that are as good as possible.

Some minimization (maximization) problems allow polynomial-time ρ-approximations ($(1/\rho)$-approximations) for $\rho = 1 + \varepsilon$ and any $\varepsilon > 0$, that is, the optimum can be

approximated arbitrarily well. An algorithm that takes as an input not only the problem instance I, but also a value $\varepsilon > 0$ and then computes a $(1+\varepsilon)$-approximation (or a $(1/(1+\varepsilon))$-approximation) for I is called a *polynomial-time approximation scheme* (PTAS) if its running time is polynomial in the size of I. The running time of a PTAS may, however, increase exponentially as ε decreases; for example, a running time of $O(n^{1/\varepsilon})$ is valid for a PTAS, where n is the size of I. If the running time of $\mathcal{A}$ is polynomial in both the size of I and $1/\varepsilon$ then $\mathcal{A}$ is called a fully polynomial-time approximation scheme (FPTAS). A valid running time for an FPTAS is, for example, $O(n^2(1/\varepsilon)^4)$.

Not all NP-hard optimization problems can be approximated equally well. There are problems for which no constant-factor approximation can exist if $\mathcal{P} \neq \mathcal{NP}$; there are also problems that allow a k-approximation for some constant k, but they cannot have a PTAS if $\mathcal{P} \neq \mathcal{NP}$. The general approach for showing hardness of approximation of an optimization problem Q is to reduce an instance I of an NP-hard decision problem R in polynomial time to an instance I' of Q such that the following holds (in the case of a minimization problem). If I is a "yes"-instance of R then $\mathrm{OPT}(I') \leq f(I')$ for some function f of the instance, and if I is a "no"-instance of R then $\mathrm{OPT}(I') > \alpha \cdot f(I')$, where α is a constant > 1 or, more generally, some function depending on the size of the instance, for example, $\log(|I'|)$. Then, since the reduction introduces a gap of α between "yes"- and "no"-instances, an α-approximation for Q could be used to decide R; hence no such α-approximation can exist if $\mathcal{P} \neq \mathcal{NP}$.

For further details and examples of approximation algorithms we refer to the textbooks by Vazirani [Vaz01] and Cormen et al. [CLRS01, Chapter 35].

2.4.2 Fixed-Parameter Algorithms

Although approximation algorithms are a promising approach for many hard problems, exact solutions are certainly preferable over approximate ones, if not even necessary in some applications. Here we present an approach for computing exact solutions to problems that are NP-hard. Clearly, this takes in general exponential (or worse) running time; the aim is, however, to find algorithms that still run efficiently for certain restricted problem instances.

The concept of parameterized algorithms for NP-hard problems is based on the idea to split off some parameter k from an input instance I of size n and then consider the running time of an algorithm in terms of n and k. The goal is to find algorithms, in which the "combinatorial explosion" is restricted to the parameter k, and the input size n appears only in polynomial terms. If k is small in typical problem instances then such an algorithm may indeed be able to find exact solutions reasonably fast.

More formally, the input instance of a parameterized problem is a tuple (I, k), where I is the actual problem instance of size n and k is a separate parameter, typically an integer. As an example for a parameterized problem, consider the problem VertexCover. A vertex cover in a graph $G = (V, E)$ is a subset $C \subseteq V$ of the vertices such that for each edge $uv \in E$ at least one endpoint is contained in C, that is, $\{u, v\} \cap C \neq \emptyset$. The problem VertexCover is then defined as follows: Given a graph $G = (V, E)$ and a non-negative integer k, is there a vertex cover $C \subseteq V$ of size $|C| \leq k$?

An algorithm $\mathcal{A}$ for such a parameterized problem is called a *fixed-parameter* or *fixed-parameter tractable* (FPT) algorithm, if the running time of $\mathcal{A}$ for the input (I, k) is $O(f(k) \cdot n^{O(1)})$, where n is the size of I and f is an arbitrary computable function independent of n. The problem VertexCover, for example, has a very simple FPT-

algorithm[3] with a worst-case running time of $O(2^k \cdot n)$; the algorithm is based on a bounded search tree technique with a search tree of size 2^k [DF98].

An FPT-algorithm is thus indeed computationally tractable for fixed k; depending on the function f, it may even be useful in practice as long as k is small enough. For further reading on parameterized complexity and fixed-parameter tractable algorithms, and in particular for an introduction to various techniques for developing efficient fixed-parameter algorithms, we refer to the monographs by Downey and Fellows [DF98] and by Niedermeier [Nie06].

2.4.3 Mathematical Programming

Mathematical programming is an umbrella term for mathematical optimization problems that are defined as follows. Given a domain S and a function $f : S \to \mathbb{R}$, find an element $x \in S$ that minimizes or maximizes f on S. In this section we consider two mathematical programming techniques: *linear programming* and *(mixed-) integer linear programming*.

Linear programming is a well-known mathematical optimization method. A linear program (LP) consists of a set of real variables and a linear objective function which is optimized subject to a set of linear constraints (equalities or inequalities) in these variables. An LP in *standard form* is written as follows:

$$\begin{array}{lrcl} \text{maximize} & c^T x & & \\ \text{subject to} & Ax & \leq & b \\ & x & \geq & 0, \end{array} \tag{2.1}$$

where $c \in \mathbb{R}^n$ is an n-dimensional vector defining the objective function, and $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ are an $m \times n$-matrix and an m-dimensional vector, respectively, which define the constraints for the n-dimensional solution vector x. Note that minimization problems, linear constraints in alternative forms, or negative variables can always be rewritten in standard form. The set $S = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ defines a (possibly unbounded) convex polytope called the *feasible region*. A vector in the feasible region is a *feasible solution* and we are interested in a feasible solution that maximizes the objective function. If the feasible region is empty, the LP is *infeasible*. As an example consider the two-dimensional (non-standard form) LP

$$\text{maximize} \qquad x + 2y \tag{2.2}$$

$$\text{subject to} \qquad -9/13 \cdot x + y \leq 11/13 \tag{2.3}$$

$$13/10 \cdot x - y \leq 9/5. \tag{2.4}$$

Each constraint of an LP defines a half space in $\mathbb{R}^n$, see the two half planes corresponding to (2.3) and (2.4) in Figure 2.4. The feasible region S, which is shaded in Figure 2.4, is the intersection of the two half planes. The feasible solutions that maximize the objective function also have a geometric interpretation. In our example the coefficient vector in (2.2) is $c = \binom{1}{2}$. If we sweep the plane in direction c with a line ℓ orthogonal to c, then the last points of S swept by ℓ are those that maximize (2.2), see point $s^\star$ in Figure 2.4. The traces of ℓ are shown as dashed lines in Figure 2.4. Linear programs can be solved efficiently, for example, using Karmarkar's interior-point method [Kar84]. Chandru and Rao give a survey [CR99b] devoted to linear programming.

[3] The currently best FPT-algorithm for VertexCover has a running time of $O(1.2852^k + kn)$ [CKJ01].

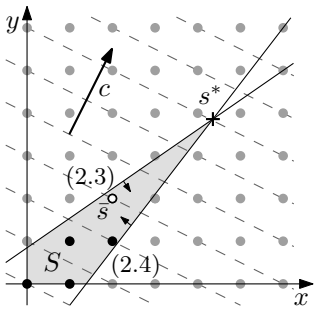


Figure 2.4: Geometric interpretation of a two-dimensional LP. The optimal fractional solution is s^* and the optimal integral solution is $\bar{s}$.

Mixed-integer programming is an extension of linear programming and allows for the use of integer variables instead of insisting on real variables, that is, in a mixed-integer program (MIP) we demand for the set of variables $\{x_1, \ldots, x_n\}$ that $x_j \in \mathbb{Z}$ for $j \in J$, where $J \subseteq \{1, \ldots, n\}$. If $J = \{1, \ldots, n\}$, that is, *all* variables are integers, the MIP is called an *integer linear program* (ILP). If we drop the integrality constraints from a MIP, we obtain a regular LP, which is called its *LP relaxation*. If we set $x, y \in \mathbb{Z}$ in the example in Figure 2.4, the integer feasible solutions consist of the grid points in S marked by black dots. Note that the optimal integer solution $\bar{s}$ is in general far from the optimal solution $s^\star$ of the LP relaxation; an optimal integer solution $\bar{s}$ can in general *not* be obtained from an optimal fractional solution $s^\star$ by simply rounding the components of the vector $s^\star$ to the next integers.

Integrality constraints make a continuous problem discrete; if the set of fractional solutions is bounded, then the number of integral solutions becomes finite. So it seems solving the more restricted problem is easier. But the opposite is the case: mixed-integer programming is NP-hard in general [GJ79]. Geometric properties of the LP that are exploited by efficient solution strategies are lost. On the other hand many hard optimization problems can be modeled as a MIP. Several successful strategies for solving MIPs have been developed, for example, *branch-and-cut*. These methods first solve the LP relaxation of a MIP and then use sophisticated branching strategies to remove fractional variables and cut off parts of the feasible region that do not contain an optimal integer solution. During this process, candidate integer solutions are computed and gradually improved. The *optimality gap* between the cost of the currently best integer solution and the lower bound given by the LP relaxation is an indicator of the solution quality. The time required to solve a MIP not only depends on its size but also strongly on how "close" the optimal integer solution is to the solution of the relaxation.

We close this section by giving an example that is a standard trick in MIP modeling and that will be useful in Chapter 3. Suppose we want to make sure that at least one of three constraints C_1, C_2, and C_3 is fulfilled, but not necessarily all of them. In other words, we want to express the disjunction $C_1 \vee C_2 \vee C_3$. Suppose

$$\begin{aligned} C_1: \quad & x - 3 \leq 0, \\ C_2: \quad & y \leq 0, \\ C_3: \quad & x + y \leq 0. \end{aligned}$$

Then we introduce three binary variables α_1, α_2, and α_3, that is, variables that are restricted to the set $\{0, 1\}$. We require that at least one of them equals 1 by the constraint

$$\alpha_1 + \alpha_2 + \alpha_3 \geq 1. \tag{2.5}$$

Now we can formulate the *disjunction* $C_1 \vee C_2 \vee C_3$ as the *conjunction* $C_1' \wedge C_2' \wedge C_3'$, where

$$\begin{array}{lrcl} C_1': & x - 3 & \leq & M(1 - \alpha_1), \\ C_2': & y & \leq & M(1 - \alpha_2), \\ C_3': & x + y & \leq & M(1 - \alpha_3), \end{array} \tag{2.6}$$

and M is a large constant that must be an upper bound on the left-hand sides of the inequalities. Note that (2.5) and (2.6) form a conjunction of linear constraints, that is, a valid part of a MIP. It is worth making M as tight a bound on the left-hand sides as possible—this helps to speed up solving the MIP.

In order to solve a MIP in practice, there are several free solvers (for example, lp_solve[4]) and commercial solvers (for example, Ilog CPLEX[5]) available. Chandru and Rao give a survey [CR99a] that contains a good overview of the theory of integer programming and of modeling discrete optimization problems as MIPs. Schrijver [Sch86] and Bertsimas and Tsitsiklis [BT97] cover theory and algorithms for (integer) linear programming in detail.

[4] see `http://lpsolve.sourceforge.net`
[5] see `http://www.ilog.com/products/cplex`

Chapter 3

Metro Maps: Layout and Labeling

A metro map is a schematic diagram of a public transport network that displays the train stations and the transport lines serving them. Metro maps are usually provided to the passengers as printed pocket maps or displayed as large posters inside metro stations and trains. They are designed as simple navigational aids to facilitate using the transport system, for example, to plan journeys and to assist passengers during their trip. Hence metro maps are optimized for readability with respect to visual route planning tasks and navigation. Until today it requires a skilled graphic designer to produce a metro map that meets all quality requirements. It is a challenging problem in network visualization to automatically draw high-quality metro maps and it has attracted several research efforts in recent years. But there is more to metro maps than just drawing the network itself. In real-world maps each station is labeled by its name. This adds a map-labeling component to the network visualization problem, which, ideally, is considered simultaneously with the network-layout problem in order to find a layout that has enough space next to the stations to place all labels without overlap.

In this chapter we present such a combined approach for drawing and labeling metro maps. Each individual problem is already NP-hard. Thus we decided to model metro map layout as a mixed-integer program (MIP) that can be optimized with standard MIP solvers. Inspired by a large number of real-world examples, we define a set of design rules for high-quality labeled metro maps. We further split these rules into hard and soft constraints. The hard constraints must be satisfied and are modeled as linear constraints in the MIP. The soft constraints must be optimized and are modeled in the linear objective function. We improve the performance of the MIP formulation using data-reduction heuristics and advanced functionalities of the MIP solver CPLEX. In three different case studies with real-world examples we show that our method can compete with most previous approaches in terms of running time and that our method yields metro-map layouts in a quality that is comparable to manually designed maps (and far better than previous approaches). The chapter is based on joint work with Alexander Wolff [NW06, NW].

3.1 Introduction

Nowadays, metro (or subway) maps are natural tools for passengers of public transport systems in large urban areas around the world. Metro maps support both commuters and foreign visitors in orienting themselves in often complex and confusing transport networks. Be it as a poster inside stations and trains or as a pocket map, their aim is to

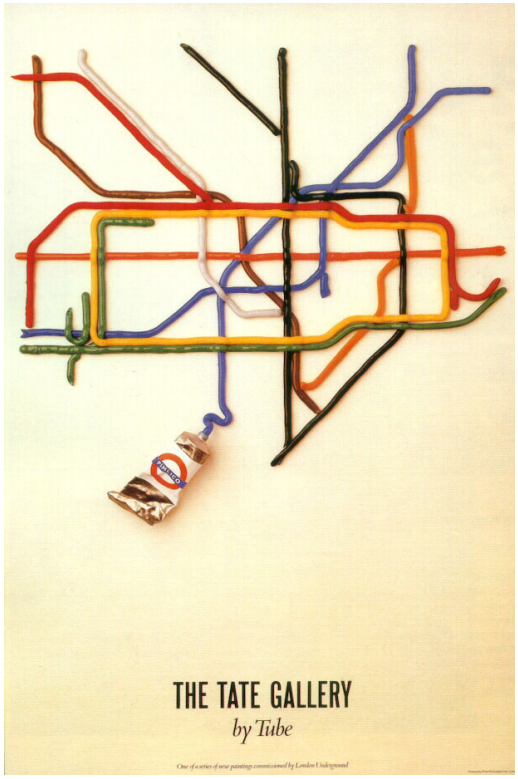


Figure 3.1: Poster advertising the Tate Gallery in London by David Booth, 1986.

help passengers to navigate in the network. One common task is visual route planning, that is, identifying on the map how to get from A to B as fast or as conveniently as possible. Once on the train, a metro map helps to answer questions like "Where do I have to change trains?", "To which line and direction do I need to transfer to?", and "How many stops remain before I must get off the train?". For this kind of questions it is not necessary to know the exact geography; it can even be hindering. Rather, it is the topology of the network that is important. This fact was first realized and exploited by Henry Beck, an engineering draftsman, who created the first schematic map of the London Underground in 1933 [Gar94]. From then on his ingenious idea spread around the globe so that today the majority of metro maps are schematic maps that follow more or less the principles of Beck's initial drafts [Ove03, Mor96]. The effectiveness of schematic public transport maps was empirically confirmed in a user study by Bartram [Bar80] that compared the route planning performance of 32 subjects using a geographic map, a schematic map, and two textual descriptions of a bus network with seven bus lines. The schematic map clearly was the best form of representing the network information for the given task. The longevity of Beck's design principles in all successive maps of the London Underground until today is another clear indication for the usefulness and the aesthetic appeal of the London Underground map. Even artists were inspired by Beck's tube map, see Figure 3.1.

Beck designed his map according to a simple set of rules: meandering transport lines are straightened and restricted to horizontals, verticals, and diagonals at 45° (we will call such a layout *octilinear*); the scale in crowded downtown areas is larger than in less dense suburbs in order to create a more uniform use of space; in spite of all distortion, the network topology and a general sense of geometry, for example, a certain relative position

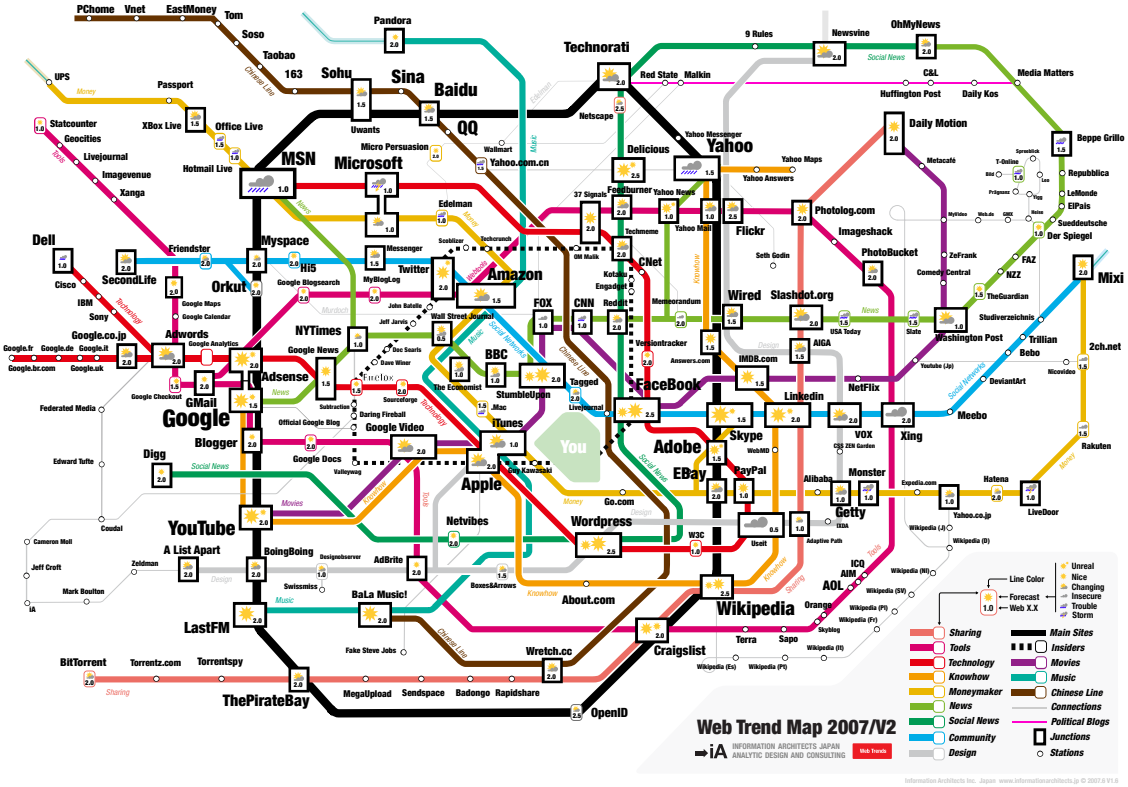


Figure 3.2: Web Trend Map 2007 by Information Architects [Inf07].

between stations, is retained. Note that a map designed according to these criteria should only be used for its intended purpose, that is, to answer navigational questions on the network. Estimating, for example, geographic distances or travel times from a metro map can be misleading.

The familiarity of many people with reading metro maps has led to the idea of using the *metro-map metaphor* to visualize abstract information without a geographic context. Sandvad et al. [SGSK01] and Nesbitt [Nes04] use the metro-map metaphor as a way to visualize guided tours in the Internet and "trains of thoughts", respectively. Stott et al. [SRB+05] present a prototype tool to draw project plans in a metro-map style. The publisher O'Reilly has used the metaphor to visualize its product lines [O'R03], and Hahn and Weinberg [HW02] draw metabolic pathways in a cancer cell as metro lines (see Figure 1.2 in Chapter 1). The Tokyo-based information design agency Information Architects published the Web Trend Map 2007 which shows the most influential web sites of the year 2007 in the style of the Tokyo metro map [Inf07]. Metro stations represent web sites, and metro lines represent different topics like technology, news, etc; a web site providing technology news is then an interchange between the technology line and the news line. Clearly, some of Beck's original layout principles need to be adapted since, for example, visualizations of abstract data usually do not have a given geometric representation.

Moreover, general octilinear graph layout, even without using the concept of metro lines, is a promising new alternative for various schematic technical and engineering drawings like cable plans, class diagrams, circuit schematics, etc. which are currently dominated by orthogonal layouts. The main benefit of octilinear layouts is that they potentially use

less space and fewer bends while still being very tidy. For example, in VLSI design the X Architecture [Tei02] is a recent effort for producing octilinear chip layouts. Another application is to compute schematic layouts of *sketches* of graphs, a concept introduced by Brandes et al. [BEKW02].

Drawing metro maps in the style of Beck can be naturally modeled as a graph drawing problem (see Section 2.2.2), where the stations of the network correspond to the set of vertices and the physical links between pairs of stations correspond to the set of edges. Accordingly, a layout algorithm for metro maps has to find positions in the plane for the vertices and edges such that the resulting drawing satisfies the basic requirements defined by the drawing conventions and optimizes a set of aesthetic criteria. Manually producing elaborate metro maps is a very costly and time-consuming process and requires a skilled graphic designer or cartographer. Thus automating the drawing of metro maps in order to assist map designers has received increasing attention in recent years by researchers in the graph drawing and information visualization communities. Avelar and Hurni [AH06] report that truly easy-to-read schematic maps exist only for few cities, mainly in North America and Western Europe. As reasons for the scarcity of good schematic maps they name a lack of funds for map preparation in the tight public transport budgets and a lack of tradition to disseminate schematic maps. Effective solutions for (semi-) automatically producing schematic public transport maps can considerably reduce the preparation cost and thus may serve as an incentive to improve existing maps or to newly introduce schematic maps. Current geographic information systems (GIS), however, do not provide automatic creation of schematic maps.

Contributions. In this chapter we propose a novel approach for automating the drawing of metro maps. We discuss related work in Section 3.2 and introduce the drawing conventions and aesthetics for metro maps in Section 3.3. Our main contribution is the translation of the metro-map layout problem problem into a mixed-integer program (MIP) in Section 3.4. The mixed-integer programming approach is—in contrast to previously suggested methods—able to distinguish between hard constraints that must be satisfied and soft constraints that are *globally* optimized. As a consequence our method is the first to model octilinearity of the resulting map as a mandatory drawing convention and not just as an aesthetic optimization criterion. We believe that octilinearity, which is strictly followed by almost all real metro maps (see [Ove03, Rob05]), is an essential ingredient for tidy and easy-to-read metro-map layouts. Furthermore, we model *label placement* for the stations as an integral part of the layout process, that is, our method reserves enough space to place all station names without overlap, see Section 3.6. This is fundamentally different from labeling a fixed drawing where in some situations labels cannot be placed without overlap due to a lack of space. The drawback of mixed-integer programming over local optimization heuristics is its potentially long running time. This is due to the fact that many NP-hard optimization problems can be modeled as a MIP which implies that mixed-integer programming is NP-hard itself. On the other hand, drawing metro maps is also NP-hard [Nöl05]. Hence, assuming $\mathcal{P} \neq \mathcal{NP}$, efficient algorithms for the problem do not exist. This justifies using MIP optimization. Furthermore, metro-map layout is an application where interactive speed is not crucial and where it is worthwhile to spend a reasonable amount of time in order to get high-quality layouts. Nonetheless, we do address the running-time issue by implementing heuristic data-reduction and speed-up methods, see Section 3.5. Section 3.7, finally, analyzes the results of our method in three case studies for the metro networks of Vienna, Sydney, and London in comparison to layouts produced by previous methods and to manually designed metro maps.

3.2 Related Work

The work on road map schematization in the following examples can be seen as a precursor of metro-map layout. Neyer [Ney99] studied a line simplification problem for polygonal paths and gave a polynomial-time algorithm to find approximations to these paths using only a restricted number of orientations. Barkowsky et al. [BLR00] used *discrete curve evolution*, an algorithm for polygonal line simplification, to draw schematic maps. As one example they looked at the lines of the Hamburg subway system. Their algorithm, however, neither restricts the edge directions nor does it increase station distances in dense downtown areas. Stations are labeled but no effort is made to avoid label overlap. Avelar and Müller [AM00, Ave07] implemented an algorithm to modify a given input map by iteratively moving the endpoints of line segments such that edges approach octilinear line segments. The algorithm was applied to the street network of Zurich [Ave08], but not to the transport network of Zurich itself (the transport lines are only superimposed on the road network). It did not quite succeed, however, in drawing all line segments octilinearly because vertex positions are calculated as arithmetic means of the best positions for a number of map constraints (for example, angle and distance constraints for each incident edge); these constraints are potentially conflicting. Cabello et al. [CdBvD$^+$01] presented an efficient algorithm for schematizing road networks. Their algorithm draws edges as octilinear paths with at most two bends and preserves the input topology. In their algorithm all vertices keep their original positions, which is in general not desired for drawing metro maps. Cabello and van Kreveld [CvK03] studied approximation algorithms for aligning points octilinearly, where each point can be placed anywhere in a locally defined region. Yet, their method does not guarantee to preserve the input topology if points correspond to vertices of a graph. Merrick and Gudmundsson [MG07] gave an algorithm for schematizing paths according to a given set of directions. They applied the algorithm to subway networks by decomposing the network into paths. Their algorithm does not guarantee, however, that the network's topology and planarity are maintained.

Jenny [Jen06] has evaluated the amount of distortion that is present in the schematic map of the London Underground by computing displacement vectors for all stations. His study illustrates the scale differences between inner city and periphery. The process of transforming two images, such as a geographically accurate and a distorted schematic map, into each other is called *image warping* in computer graphics. Böttger et al. [BBDZ08] recently presented a dynamic map warping method called *warping zoom* that combines zooming and warping. Their method continuously warps between a schematic map with distorted geography at smaller scales and a geographically accurate map at larger scales. Such a dynamic map that combines a geographic and a schematic map has the advantage that there is no loss of context in comparison to using two separate maps.

There are basically two previous approaches concerned exclusively with the automation of drawing metro maps; see also the survey article of Wolff [Wol07] about the state of the art in metro-map drawing. The first approach is based on the spring-embedder paradigm that has been introduced in Section 2.2.2. Initially, Lauther and Stübinger [LS02] sketched a system that uses a spring embedder to draw orthogonal schematic cable plans. It can be seen as a precursor of the work of Hong et al. [HMdN06] who adapted a topology-maintaining spring embedder to the special requirements of metro maps. Their method realizes edges as straight-line segments and takes edge weights into account as target edge lengths. These edge weights are determined in a preprocessing step that simplifies the input graph by collapsing all degree-2 vertices: the weight of each edge in the simplified graph corresponds

to the number of original edges that it represents. Modeling octilinearity is achieved by magnetic forces that drag each edge towards its closest octilinear direction. The geometry of the input network is considered only implicitly by optionally using the original embedding as initial layout. Having computed the final layout, all degree-2 vertices are re-inserted on the corresponding edges in an equidistant manner. Station labels are placed in an independent second step by an interactive map labeling system called LabelHints [dNE08], which avoids label–label overlaps while label–edge overlaps are not considered.

Stott and Rodgers [SR04] draw metro maps using multi-criteria optimization based on hill climbing. For a given layout they define metrics for evaluating the number of edge intersections, the octilinearity and length of edges, the angular resolution at vertices, and the straightness of metro lines. They define the quality of a layout to be a weighted sum over these five metrics. Iteratively, they consider alternative grid positions for each vertex starting with the geographic layout. Only vertex positions that preserve the topology and improve the quality measure are accepted. The authors observe that the algorithm gets stuck in local minima which is a typical drawback of local optimization techniques. They give a heuristic fix that overcomes one class of such problems and use a similar optional edge contraction step as Hong et al. [HMdN06] to preprocess the input graph. Subsequently, Stott and Rodgers [SR05] extended their previous method by integrating horizontal station labeling into the optimization process. For a given labeling they defined several criteria to evaluate the labeling quality. These criteria measure the number of occlusions of vertices, edges, and other labels, the position of the label with respect to its vertex, side consistency for labels on a path between two interchanges, and proximity to unrelated vertices. After each iteration of vertex movements there is a label-placement iteration in which the best of eight admissible label positions is selected for each vertex. The authors experienced occasional label–label overlaps, especially along horizontal edges. In his PhD thesis [Sto08] Stott weakened this effect by introducing line breaks for long station names.

Mixed-integer programming has been used occasionally in graph drawing before. Jünger and Mutzel [JM97] were the first to use integer linear programming (ILP) for a combinatorial two-layer crossing minimization problem. Klau and Mutzel [KM99b] gave an ILP formulation for the compaction phase in the topology-shape-metrics framework that minimizes the total edge length of the drawing subject to shape constraints depending on a given orthogonal representation. They extended their model with the placement of non-overlapping vertex labels in the compaction phase [KM99a]. Binucci et al. [BDLN05] gave a MIP formulation to minimize the area in the compaction phase in the presence of vertex and edge labels.

3.3 Model

What are the characteristic properties of a metro map? In order to define the metro-map layout problem in graph-drawing terms problem we need to find the drawing conventions, aesthetics, and constraints that distinguish a metro map. Although the layout principles of real metro maps differ from city to city there are some basic design rules to which almost all schematic metro maps adhere and that date back to the first tube maps designed by Beck [Gar94]. Avelar and Hurni [AH06] described some guidelines for designing schematic maps, but their focus is rather on the rendering problem (use of colors, symbols for points, background features to retain) and less on the layout of the transport network itself.

Nonetheless, Avelar and Hurni mentioned octilinearity, topological correctness, and edge straightening as characteristics of schematic transport maps.

3.3.1 Design Rules

After studying the layout principles of a large number of official metro maps [Ove03, Rob05] we identified the following design rules for the network layout of metro maps, some of which (or slight variations thereof) have also been described before [AH06, HMdN06, SR04]:

(R1) Restrict all line segments to the four *octilinear* orientations[1] horizontal, vertical, and 45°-diagonal.

(R2) Do not change the geographical network topology. This is crucial to support the mental map of the passengers.

(R3) Avoid bends along individual metro lines, especially in interchange stations, to keep them easy to follow for map readers. If bends cannot be avoided, obtuse angles are preferred over acute angles.

(R4) Preserve the relative position between stations to avoid confusion with the mental map. For example, a station being north of some other station in reality should not be placed south of it in the metro map.

(R5) Keep edge lengths between adjacent stations as uniform as possible with a strict minimum length. This usually implies enlarging the city center at the expense of the periphery.

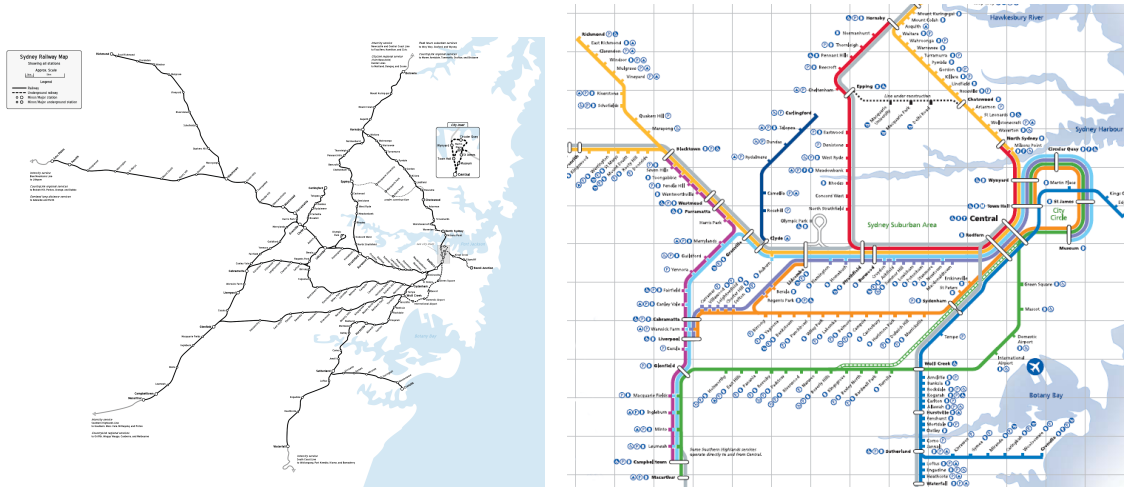
(R6) Stations must be labeled and station names should not obscure other labels or parts of the network. Horizontal labels are preferred, and labels along the track between two interchanges should use the same side of the corresponding path if possible.

(R7) Use distinctive colors to denote the different metro lines. This means that edges used by multiple lines are drawn thicker and use colored copies for each line.

As an example of how these seven rules are implemented in practice, we consider the Sydney CityRail network that is shown in Figure 3.3. This network is also used later on in our case study in Section 3.7.2 as a benchmark example since it has been used by Hong et al. [HMdN06] and by Stott and Rodgers [SR04] before. Figure 3.3a shows the geographic layout of the suburban part of the CityRail network; Figure 3.3b shows the corresponding clipping of the official network map drawn by professional graphic designers [Syd08]. Note how the seven design rules are realized in this map: all lines are octilinear (R1), the topology is preserved (R2) (hard to see in the city circle to the right of the map—a good example where non-uniform map scale is used), unnecessary bends are (mostly) avoided (R3), the mental map is retained (R4), edge lengths are rather uniform (R5), labels are non-overlapping (R6), and distinct colors are used for the individual lines (R7).

Clearly, each metro map can only be a compromise of the above criteria. For example, a map with a minimum number of line bends may drastically distort the mental map and, conversely, strictly preserving the mental map may require a large number of bends.

[1] Each of the four orientations has two directions, thus the term *octi*linear.



(a) Geographic layout.

(b) Corresponding clipping of the official map [Syd08].

Figure 3.3: The Sydney CityRail network.

3.3.2 Formal Model

We now state the metro-map layout problem in graph drawing terms. Let $G = (V, E)$ be a plane graph. We assume that we know the geographic position $\Pi(v)$ of each vertex $v \in V$ in the plane. Note that if the input layout of G contains edge crossings, we obtain a plane graph G' by introducing dummy vertices that represent the crossings. These will be preserved by the layout algorithm. As usual n and m denote the number of vertices and edges of G, respectively. Let $\mathcal{L}$ be a *line cover* of G, that is, a set of paths of G such that each edge of G belongs to at least one element of $\mathcal{L}$. An element $\ell \in \mathcal{L}$ is called a *line* and corresponds to a metro line of the underlying transport network. Let N denote the *total edge size* of $\mathcal{L}$, that is, $N = \sum_{\ell \in \mathcal{L}} |\ell|$, where $|\ell|$ is the number of edges of line ℓ. We denote the pair $(G, \mathcal{L})$ as the *metro graph*.

The task is now to find a drawing Γ of $(G, \mathcal{L})$ according to the rules (R1)–(R7). At this point we ignore rule (R7) which affects only the way Γ is displayed in the end. Furthermore we postpone the label placement given by rule (R6) to Section 3.6 and concentrate on rules (R1)–(R5). Some of these rules formulate strict requirements while others are optimization criteria. Thus we split the remaining requirements into mandatory global drawing conventions and local constraints, also called *hard constraints*, and aesthetics, also called *soft constraints*, to be optimized. Our hard constraints are:

(H1) For each edge e, the line segment $\Gamma(e)$ must be octilinear.

(H2) For each vertex v, the circular order of its neighbors must agree in Γ and the input embedding.

(H3) For each edge e, the line segment $\Gamma(e)$ must have length at least l_e.

(H4) Each edge e must have distance at least $d_{\min} > 0$ from each non-incident edge in Γ.

Constraint (H1) models octilinearity (R1), (H2) models the topology requirement (R2), (H3) models the minimum edge length in (R5), and (H4) forbids edge crossings and thus also models a part of (R2). This is because two intersecting edges would have distance $0 < d_{\min}$.

The soft constraints are intended to hold as tightly as possible. They determine the aesthetic quality of Γ and are as follows:

(S1) The lines in $\mathcal{L}$ should have few (and obtuse) bends in Γ.

(S2) For each pair of adjacent vertices (u, v), their relative position should be preserved, that is, the angle $\angle(\Gamma(u), \Gamma(v))$ should be similar to the angle $\angle(\Pi(u), \Pi(v))$, where $\angle(a, b)$ is the angle between the x-axis in positive direction and the line through a and b directed from a to b.

(S3) The total edge length of Γ should be small.

Clearly, constraint (S1) models minimizing the bends (R3) and (S2) models preserving the relative position (R4). The uniform edge length rule (R5) is realized by the combination of a strict lower bound of unit length (H3) and a soft upper bound (S3) for the edge lengths. Rule (R4) for the relative position can be interpreted as both a soft and a hard constraint, for example, one may restrict the angular deviation to at most 90° as a hard constraint and charge costs for smaller deviations as a soft constraint. Our framework reflects this ambivalence, but modeling relative position as a purely soft constraint is also possible. Of course, other soft constraints can be added or removed depending on the application. The soft constraints can be weighted according to their importance. We now formally state the metro-map layout problem.

Problem 3.1 (Metro-Map Layout Problem) *Given a plane graph $G = (V, E)$ with maximum degree 8 and vertex coordinates in $\mathbb{R}^2$, a line cover $\mathcal{L}$ of G, minimum edge lengths $l_e > 0$ for each $e \in E$, and a minimum distance $d_{min} > 0$, find a* nice *drawing Γ of $(G, \mathcal{L})$, that is, a drawing Γ that satisfies the hard constraints (H1)–(H4) and optimizes the soft constraints (S1)–(S3).*

Note that the restriction to graphs with maximum vertex degree 8 is an immediate consequence of the restriction to octilinear edge directions. Recall the difference between edges and lines in our model: while a vertex can have at most eight incident edges there can still be multiple lines that share a single edge. We are not aware of any real metro map that has vertices with a degree higher than 8 in the underlying graph.

From a theoretical point of view one can ask the existence question "Given the input, is there a drawing that satisfies all hard constraints?" It has been proven that this question is NP-hard [Nöl05]; the proof is by reduction from the Planar3-Sat problem. This result is in sharp contrast to the orthogonal setting where the same existence question can be answered by an efficient network-flow algorithm in the topology-shape-metrics framework [Tam87, GT96].

If we combine graph drawing and labeling, the only difference to Problem 3.1 is that we have an additional hard constraint that must be satisfied:

(H5) Each vertex $v \in V$ is labeled by a horizontal or diagonal label that does not occlude any other label, vertex, or edge in the drawing Γ.

For ease of presentation, we first consider Problem 3.1 to find an unlabeled drawing of G in Section 3.4. Afterwards, in Section 3.6, we present the extensions to our model that are necessary in order to satisfy constraint (H5) and to solve the combined metro map layout and labeling problem.

3.4 Mixed-Integer Programming

We decided to formulate the metro-map layout problem as a mixed-integer program. Solving NP-hard optimization problems like ours with a MIP formulation is different from using heuristic search methods like force models [HMdN06] or hill climbing [SR04]. Unlike heuristic methods, MIP takes a global approach and guarantees to find an optimal solution, albeit not in polynomial time. A clear advantage of MIP is that there are sophisticated and versatile solvers available, which means that a MIP model can be quickly implemented and tested. The main challenge is thus to formulate a MIP model that correctly and efficiently reflects the optimization problem. In this section, we show how to transform the hard and soft constraints (H1)–(H4) and (S1)–(S3) into the linear (in-) equalities and the linear objective function of a MIP. If a layout that conforms to all hard constraints exists (this was the case in all our examples), then our MIP finds such a layout. Otherwise the solver reports infeasibility. Moreover, our MIP optimizes the weighted sum of cost functions each of which corresponds to a soft constraint.

3.4.1 Coordinate System and Metric

We can state all our constraints using Cartesian coordinates. Still, for simplicity we use an extended (x, y, z_1, z_2)-coordinate system that allows us to handle all four orientations in the same way. Each coordinate axis corresponds to one of the four octilinear orientations, see Figure 3.4. For a vertex $v \in V$ we define $z_1(v) = (x(v)+y(v))/2$ and $z_2(v) = (x(v)-y(v))/2$.

Furthermore, we need to specify an underlying metric for measuring distances. We use the L^∞-metric, which defines the distance of two vertices u and v to be $\max(|x(u) - x(v)|, |y(u) - y(v)|)$. This metric has the property that all points on the boundary of the unit square centered at a point p have the same distance from p. In Figure 3.4, the eight grid points on the octilinear coordinate axes at unit L^∞-distance from the origin are marked by small circles. A side-effect of using the L^∞-metric is that all vertices are placed on a rectilinear grid as long as all edge lengths in the L^∞-metric are integers.

3.4.2 Octilinearity and Edge Length (H1) & (H3)

The constraints in this part deal with the orientation and the length of all edges $uv \in E$ and thus model the two hard constraints (H1) and (H3). In general, each edge can take any of the eight octilinear directions. With the relative position rule (R4) in mind, we further restrict the admissible directions for an edge uv to the three closest octilinear approximations of the input line segment $\overline{\Pi(u)\Pi(v)}$. This means that the maximum deviation of the angles $\angle(\Gamma(u), \Gamma(v))$ and $\angle(\Pi(u), \Pi(v))$ is 67.5°. This restriction is optional and could just as well be extended to more than three admissible directions or dropped completely. The more directions we admit the larger the solution space of the MIP gets and hence the longer it takes to solve it. It turned out that allowing three directions is a good balance of layout flexibility and solution speed.

Before formulating the constraints we need some notation to address relative positions between vertices and to denote directions of edges. For technical reasons we direct all edges arbitrarily and assume that an edge $e = uv$ is directed from u to v. For each vertex u we define a partition of the plane into eight sectors. Each sector is a 45°-wedge with apex u. The wedges are centered around rays that emanate from u and follow the eight octilinear directions. The sectors are numbered from 0 to 7 counterclockwise starting with the positive x-direction, see Figure 3.5.

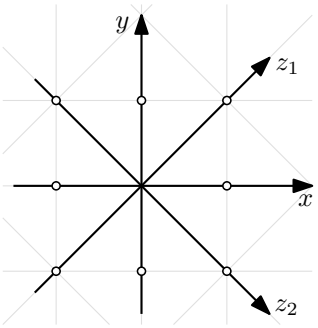


Figure 3.4: Octilinear coordinate system. Marked grid points have unit L^∞-distance from the origin.

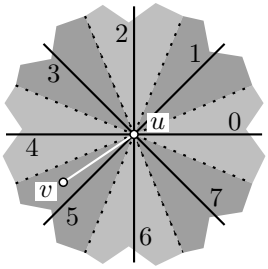


Figure 3.5: Numbering of the sectors and the octilinear directions relative to vertex u, for example $\mathrm{sec}_u(v) = 5$.

To denote the rough relative position between two vertices u and v in the *original layout* we use the terms $\mathrm{sec}_u(v)$ and $\mathrm{sec}_v(u)$ representing the sector relative to u in which v lies and the sector relative to v in which u lies, respectively. These values describe the known input layout. Similarly, for each edge uv, we define integer variables $\mathrm{dir}(u,v)$ and $\mathrm{dir}(v,u)$ to denote the octilinear direction of uv in the *new layout*, that is, these variables describe the unknown output layout. We identify each octilinear direction with its corresponding sector. For example, if the edge uv leaves u in negative z_1-direction, we say $\mathrm{dir}(u,v) = 5$. Both values are symmetric to a reversal of the viewpoint. So considering an edge uv in reversed direction vu corresponds to shifting the sector origin from u to v. Then the sector of v with respect to the origin u is opposite of the sector of u with respect to origin v (see Figure 3.5) or, equivalently, we can add 4 to the sector value and the direction variable, that is, $\mathrm{sec}_u(v) = \mathrm{sec}_v(u) + 4 \pmod 8$ and $\mathrm{dir}(u,v) = \mathrm{dir}(v,u) + 4 \pmod 8$.

The following three blocks of constraints model the layout of the edge uv:

$$\alpha_{\mathrm{prec}}(u,v) + \alpha_{\mathrm{orig}}(u,v) + \alpha_{\mathrm{succ}}(u,v) \;=\; 1 \tag{3.1}$$

$$\begin{aligned} \mathrm{dir}(u,v) &= \textstyle\sum_{i\in\{\mathrm{prec,orig,succ}\}} \mathrm{sec}_u^i(v) \cdot \alpha_i(u,v) \\ \mathrm{dir}(v,u) &= \textstyle\sum_{i\in\{\mathrm{prec,orig,succ}\}} \mathrm{sec}_v^i(u) \cdot \alpha_i(u,v) \end{aligned} \tag{3.2}$$

$$\begin{aligned} y(u) - y(v) &\le M(1-\alpha_{\mathrm{prec}}(u,v)) \\ -y(u) + y(v) &\le M(1-\alpha_{\mathrm{prec}}(u,v)) \\ x(u) - x(v) &\ge -M(1-\alpha_{\mathrm{prec}}(u,v)) + l_{uv}. \\ &\vdots \end{aligned} \tag{3.3}$$

Constraint (3.1) models the selection of one of the three permitted directions by means of three binary variables α_{prec}, α_{orig}, α_{succ} whose sum equals 1. The index $i \in \{\mathrm{prec,orig,succ}\}$ for which $\alpha_i(u,v) = 1$ denotes the direction of the original sector $\mathrm{sec}_u(v)$ of edge uv ($i = \mathrm{orig}$), its preceding sector ($i = \mathrm{prec}$), or its succeeding sector ($i = \mathrm{succ}$), respectively. By $\mathrm{sec}_u^i(v)$ we denote the index of these sectors for $i \in \{\mathrm{prec,orig,succ}\}$. In the example of Figure 3.5 these are sectors 4, 5, and 6.

In the constraints (3.2) the integer variables $\mathrm{dir}(u,v)$ and $\mathrm{dir}(v,u)$ are assigned to the correct edge direction numbers according to the assignment of the three binary variables above. The direction variables will be used in some of the remaining hard and soft constraints.

Finally, the constraints (3.3) deal with the actual positions of vertices u and v. For each possible direction we need such a set of three inequalities, which of course depend on the direction. Only the set of constraints corresponding to the selected direction shall be active and this disjunctive condition is modeled by means of a (large) constant M as introduced in Section 2.4.3. Constraints (3.3) are an example where the preceding sector $\text{sec}_u^{\text{prec}}(v)$ equals 4 as in Figure 3.5, that is, uv should be directed horizontally to the left. In this case v should have the same y-coordinate as u and lie at least l_{uv}, the minimum length of uv, to the left of u. Exactly this requirement is modeled by constraints (3.3) if $\alpha_{\text{prec}}(u, v) = 1$. Otherwise, if $\alpha_{\text{prec}}(u, v) = 0$, all constraints are trivially satisfied since M is chosen as an upper bound on all possible coordinate differences, for example, if $0 \leq x(v), y(v) \leq n$ for all $v \in V$ then $M = n$ would suffice. The sets of constraints are similar for other directions and for $i = \text{orig}$ or $i = \text{succ}$, respectively: one coordinate of u and v must be equal and their distance along the respective octilinear direction must be at least the minimum edge length l_{uv}.

Overall, the above constraints model octilinearity (H1) and the lower bound on the length of each edge (H3). Clearly, the number of possible directions can be increased in the above formulation if the relative position rule (R4) for adjacent vertices is not modeled as a partially hard constraint. The restriction to three directions is a good compromise between conservation of the relative position and flexibility in the drawing.

Each edge of G gives rise to 5 variables and 12 constraints; this totals $5m$ variables and $12m$ constraints.

3.4.3 Circular Vertex Orders (H2)

The constraints in this part preserve the circular order of the neighbors around each vertex and thus the input embedding as required by hard constraint (H2). For each vertex v with $\deg(v) \geq 2$ we introduce the following constraints:

$$\beta_1(v) + \beta_2(v) + \ldots + \beta_{\deg(v)}(v) = 1 \tag{3.4}$$

$$\begin{aligned} \text{dir}(v, u_1) &\leq \text{dir}(v, u_2) - 1 + 8\beta_1(v) \\ \text{dir}(v, u_2) &\leq \text{dir}(v, u_3) - 1 + 8\beta_2(v) \\ &\vdots \\ \text{dir}(v, u_{\deg(v)}) &\leq \text{dir}(v, u_1) - 1 + 8\beta_{\deg(v)}(v), \end{aligned} \tag{3.5}$$

where $\beta_i(v)$ are binary variables for $i = 1, \ldots, \deg(v)$ and $u_1 < \ldots < u_{\deg(v)}$ are the neighbors of v in counterclockwise order with respect to the input embedding.

The idea behind (3.4) and (3.5) is that the circular input order of the incident edges of v must be reflected by the values of the direction variables $\text{dir}(v, u_1), \ldots, \text{dir}(v, u_{\deg(v)})$. Thus looking at the edges in the given order, their direction index must strictly increase except for one position. Namely, it decreases when we cross the boundary between sector 7 and sector 0. Hence there is exactly one of the inequalities $\text{dir}(v, u_i) \leq \text{dir}(v, u_{i+1}) - 1$ that does not hold unless we add 8 to the right-hand side. The position i where this happens is determined by the only binary variable in constraint (3.4) with $\beta_i(v) = 1$. For this i, the corresponding constraint in (3.5) evaluates to $\text{dir}(v, u_i) \leq \text{dir}(v, u_{i+1}) - 1 + 8$ which holds even if $\text{dir}(v, u_i) > \text{dir}(v, u_{i+1}) - 1$. All other constraints for $j \neq i$ in (3.5) do not add 8 to the right-hand side as $\beta_j(v) = 0$ by constraint (3.4).

Note that we demand strictly increasing direction indices and thus no two edges incident to the same vertex can have the same direction. For each vertex v this part of the MIP

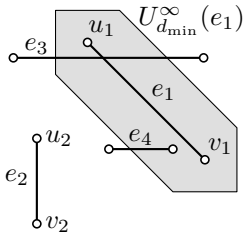


Figure 3.6: The $d_{\min}$-neighborhood of e_1; e_2 satisfies (H4) with respect to e_1, but e_3 and e_4 do not.

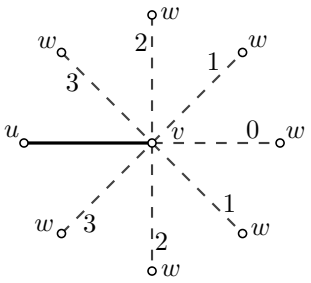


Figure 3.7: Bend cost $\text{bend}(u, v, w)$ for each value of $\text{dir}(v, w)$.

requires $\deg(v)$ binary variables and $\deg(v) + 1$ constraints. Since $\sum_{v \in V} \deg(v) = 2m$ this totals $2m$ variables and $2m + n$ constraints.

3.4.4 Edge Spacing (H4)

As stated before, constraint (H4), which requires that two non-incident edges stay $d_{\min}$ apart, avoids that edge crossings are introduced and thus ensures the planarity of the drawing. For each pair of non-incident edges $(e_1, e_2) = (u_1v_1, u_2v_2)$ we have:

$$\sum_{i \in \{\text{N,S,E,W,NE,NW,SE,SW}\}} \gamma_i(e_1, e_2) \geq 1 \tag{3.6}$$

$$\begin{aligned} x(u_2) - x(u_1) &\leq M(1 - \gamma_E(e_1, e_2)) - d_{\min} \\ x(u_2) - x(v_1) &\leq M(1 - \gamma_E(e_1, e_2)) - d_{\min} \\ x(v_2) - x(u_1) &\leq M(1 - \gamma_E(e_1, e_2)) - d_{\min} \\ x(v_2) - x(v_1) &\leq M(1 - \gamma_E(e_1, e_2)) - d_{\min}, \\ &\vdots \end{aligned} \tag{3.7}$$

where $\gamma_{\text{N}}(e_1, e_2), \ldots, \gamma_{\text{SW}}(e_1, e_2)$ are binary variables. The idea behind these constraints is that for a pair of octilinear edges to have L^{∞} distance of at least $d_{\min}$ it suffices to ensure that the two edges stay apart by $d_{\min}$ in at least one of the eight octilinear directions, which we denote by the compass directions N, S, E, W, NE, NW, SE, and SW. Figure 3.6 shows the $d_{\min}$-neighborhood $U^{\infty}_{d_{\min}}(e_1)$ of an edge e_1. To make sure that no other edge intersects $U^{\infty}_{d_{\min}}(e_1)$ we enforce that both vertices of that edge have a distance of at least $d_{\min}$ from e_1 in the *same* octilinear direction—unlike the edges e_3 and e_4 in Figure 3.6.

From constraint (3.6) we get that at least one variable $\gamma_i(e_1, e_2)$ equals 1. Let for instance $\gamma_{\text{E}}(e_1, e_2) = 1$, that is, e_1 is east of e_2 as in the example in Figure 3.6. The corresponding block of constraints for $\gamma_{\text{E}}(e_1, e_2)$ is given in (3.7); for the other seven variables there are similar sets of constraints. Since $\gamma_{\text{E}}(e_1, e_2) = 1$, the four constraints in (3.7) simply mean that both u_2 and v_2 must be at least $d_{\min}$ to the left of both u_1 and u_2. Otherwise, if $\gamma_{\text{E}}(e_1, e_2) = 0$, the inequalities are always satisfied. The same principles apply for the constraints of the remaining orientations.

For each pair of edges we thus need 33 constraints and eight binary variables. Since there are $\Theta(m^2)$ such pairs, however, the constraints and variables that model (H4) dominate the otherwise linear size of our model. This slows down the MIP solution time drastically.

In Section 3.5.2 we propose two (heuristic) improvements to the model that significantly cut down the number of constraints and variables for modeling (H4).

Also note that the above planarity constraints are based on the fact that, due to a limited number of edge directions, there is only a limited number of relative positions of two edges. This model therefore does not extend to planarity of arbitrary line segments.

3.4.5 Line Bends (S1)

Usability of a metro map depends strongly on the user's ability to visually follow its metro lines. This is usually facilitated by using distinguishable colors (see rule (R7)), but also by avoiding bends along the lines as formulated in (S1).

We define the bend cost $\text{bend}(u, v, w)$ subject to the actual angle between two adjacent edges uv and vw on a path $\ell \in \mathcal{L}$. Due to the octilinearity constraints and to the fact that two adjacent edges cannot have the same direction relative to their joint vertex, the angles can only equal 180°, 135°, 90°, and 45°. In that order we define the corresponding bend cost to be 0, 1, 2, and 3; the cost increases with the acuteness of the angle, see Figure 3.7.

Then the total bend cost of the drawing is

$$\text{cost}_{(S1)} = \sum_{\ell \in \mathcal{L}} \sum_{uv,vw \in L} \text{bend}(u, v, w). \tag{3.8}$$

Minimizing $\text{cost}_{(S1)}$ hence minimizes the number and acuteness of the bends along all lines in $\mathcal{L}$. We may assign higher, for example, double, costs to bends in interchange vertices to stress that lines should go straight through these vertices.

It remains to state how the bend cost $\text{bend}(u, v, w)$ is actually computed within the model. We can determine the angle between two adjacent edges uv and vw by reusing the values of $\text{dir}(u, v)$ and $\text{dir}(v, w)$ that have been defined in Section 3.4.2. For ease of notation let $\Delta\text{dir}_{u,v,w} = \text{dir}(u, v) - \text{dir}(v, w)$. It is easy to verify that the bend cost defined above can be expressed as

$$\text{bend}(u, v, w) = \min\{|\Delta\text{dir}_{u,v,w}|,\ 8 - |\Delta\text{dir}_{u,v,w}|\}, \tag{3.9}$$

where the first term is minimum for $-4 \leq \Delta\text{dir}_{u,v,w} \leq 4$ and the latter term for $-7 \leq \Delta\text{dir}_{u,v,w} \leq -5$ or $5 \leq \Delta\text{dir}_{u,v,w} \leq 7$. In order to compute this cost by means of linear constraints we use

$$\begin{aligned} -\text{bend}(u, v, w) &\leq \Delta\text{dir}_{u,v,w} - 8\delta_1(u, v, w) + 8\delta_2(u, v, w) \\ \text{bend}(u, v, w) &\geq \Delta\text{dir}_{u,v,w} - 8\delta_1(u, v, w) + 8\delta_2(u, v, w), \end{aligned} \tag{3.10}$$

where $\delta_1(u, v, w)$ and $\delta_2(u, v, w)$ are binary variables. These constraints express that $\text{bend}(u, v, w)$ is lower bounded by $|\Delta\text{dir}_{u,v,w} - 8\delta_1(u, v, w) + 8\delta_2(u, v, w)|$. Since $\text{bend}(u, v, w)$ is minimized in $\text{cost}_{(S1)}$, it will match its lower bound. Moreover, as a result of this minimization, the lower bound will itself be minimized by assigning the best possible values to the two binary variables $\delta_1(u, v, w)$ and $\delta_2(u, v, w)$. For $5 \leq \Delta\text{dir}_{u,v,w} \leq 7$ setting $\delta_1(u, v, w) = 1$ and $\delta_2(u, v, w) = 0$ yields the smallest value; for $-7 \leq \Delta\text{dir}_{u,v,w} \leq -5$ setting $\delta_1(u, v, w) = 0$ and $\delta_2(u, v, w) = 1$ yields the smallest value; in the remaining cases either both variables are set to one or to zero. In all these cases we have $|\Delta\text{dir}_{u,v,w} - 8\delta_1(u, v, w) + 8\delta_2(u, v, w)| = \min\{|\Delta\text{dir}_{u,v,w}|,\ 8 - |\Delta\text{dir}_{u,v,w}|\}$ as desired.

Minimizing the number of bends thus uses three variables and two constraints for each pair of incident edges on a path $\ell \in \mathcal{L}$. Since there are in total at most N (the total edge size of $\mathcal{L}$) such pairs we are using at most $3N$ variables and at most $2N$ constraints.

3.4.6 Relative Positions (S2)

To preserve as much of the overall appearance of the input geometry of the metro system as possible we have already restricted the edge directions to the set of the three octilinear directions closest to the input direction of each edge in Section 3.4.2. Ideally, we want to draw each edge uv using its best octilinear approximation, that is, the direction where $\mathrm{dir}(u,v) = \mathrm{sec}_u(v)$. We introduce a cost of 1 if the layout does not use that direction. This suffices to model (S2) in our case. In the general case, in which more than three directions are admissible, a gradual cost scheme similar to the bend cost above could be applied.

For each edge uv we define as its cost a binary variable $\mathrm{rpos}(uv)$ which can be set to zero if and only if $\mathrm{dir}(u,v) = \mathrm{sec}_u(v)$. Then the cost for deviating from the original relative positions is

$$\mathrm{cost}_{(S2)} = \sum_{uv \in E} \mathrm{rpos}(uv) \tag{3.11}$$

which, for each edge, charges 1 if not using the nearest octilinear direction. Due to the minimization of $\mathrm{cost}_{(S2)}$ the correct assignment of $\mathrm{rpos}(uv)$ is modeled by

$$-8 \cdot \mathrm{rpos}(uv) \leq \mathrm{dir}(u,v) - \mathrm{sec}_u(v) \leq 8 \cdot \mathrm{rpos}(uv), \tag{3.12}$$

where 8 is a trivial bound on the absolute value of the considered difference of directions. This part of the model needs m variables and $2m$ constraints.

3.4.7 Total Edge Length (S3)

The edge lengths are considered in the L^∞-metric as stated before. We define a new real-valued, non-negative variable $\mathrm{len}(uv)$ for each edge uv that serves as an upper bound on the length of uv. By minimizing the sum of all upper bounds

$$\mathrm{cost}_{(S3)} = \sum_{uv \in E} \mathrm{len}(uv), \tag{3.13}$$

the bounds $\mathrm{len}(uv)$ become tight and thus indeed equal to the corresponding edge lengths. The constraints that define $\mathrm{len}(uv)$ are simply

$$\begin{aligned} x(u) - x(v) &\leq \mathrm{len}(uv) \\ -x(u) + x(v) &\leq \mathrm{len}(uv) \\ y(u) - y(v) &\leq \mathrm{len}(uv) \\ -y(u) + y(v) &\leq \mathrm{len}(uv). \end{aligned} \tag{3.14}$$

In total we use m variables and $4m$ constraints for modeling (S3).

3.4.8 Summary of the Model

In the previous seven subsections we have described in detail the constraints and variables of our MIP model for the metro-map layout problem. The hard constraints (H1)–(H4) form the constraint section of the MIP. The soft constraints (S1)–(S3) contribute another part to the constraint section that defines the cost variables. The cost variables are subsequently minimized in the (weighted) objective function

$$\lambda_{(S1)}\mathrm{cost}_{(S1)} + \lambda_{(S2)}\mathrm{cost}_{(S2)} + \lambda_{(S3)}\mathrm{cost}_{(S3)}, \tag{3.15}$$

constraint	# MIP variables	# MIP constraints
(H1) & (H3)	$5m$	$12m$
(H2)	$2m$	$2m + n$
(H4)	$\leq 8(m^2 - m)/2$	$\leq 33(m^2 - m)/2$
(S1)	$3N$	$2N$
(S2)	m	$2m$
(S3)	m	$4m$
total	$\leq 4m^2 + 5m + 3N$	$\leq 16.5m^2 + 3.5m + 2N + n$

Table 3.1: Number of variables and constraints for each hard and soft constraint in the model. Note that (H4) applies only to non-incident edge pairs, whose number depends on the input. Hence we give only upper bounds for (H4).

where $\lambda_{(S1)}$, $\lambda_{(S2)}$, and $\lambda_{(S3)}$ are a non-negative weights that allow for adjustment of the relative importance of each of the optimization criteria.

Table 3.1 summarizes the number of variables and constraints required for each part of our model. Note that with $\Theta(m^2)$ variables and constraints (H4) is by far the most expensive part of the model, which otherwise has only linear size.

3.5 Reduction of the Problem Size

The size of the MIP model as presented in the previous section is a major factor affecting the required solution time. There are two ways to reduce the size without compromising layout quality. The first approach reduces the size of the input graph before it is translated into the MIP model. The second approach tries to identify the constraints that are relevant for the solution and hides the irrelevant constraints from the optimizer.

3.5.1 Reduction of the Network Size

A common feature of metro graphs is that they tend to have a large number of degree-2 vertices, which represent the non-interchange stations between two interchanges. By soft constraint (S1) it is desirable to avoid line bends in these degree-2 vertices, and optimizing each edge on a path between two interchanges individually seems unnecessary. Therefore, the idea to replace each path of degree-2 vertices temporarily by a single edge (which will always be drawn straight) and to reinsert the vertices in the final drawing equidistantly on this edge has been proposed in the literature [HMdN06, SR04]. We use a slightly different approach that allows more flexibility in the layout of paths of degree-2 vertices: instead of a single edge we replace each such path by a path of length 3 that can thus have up to two bends between two neighboring interchanges. This allows for better balancing line straightness (S1) and geographic accuracy (S2) in the layout. Again, the original vertices are reinserted equidistantly on their corresponding paths. Our case studies in Section 3.7 show that this is indeed a good compromise between layout flexibility and the reducing the size of the MIP model.

3.5.2 Reduction of the MIP Size

The time that is required to solve a MIP depends on the geometric shape of the feasible region, which in turn depends on the number of variables and constraints of the model. Thus reducing the model size is another way of speeding up our layout method.

As can be seen from Table 3.1, edge spacing (H4), which is also responsible for avoiding edge crossings, is the only layout constraint that causes a quadratic number of variables and constraints in the model. This is due to the fact that naively we consider (H4) for all $\Theta(m^2)$ pairs of non-incident edges. The first observation is that for a planar drawing of an embedded graph it suffices to require that all pairs of non-incident edges that are incident to the *same* face satisfy (H4). The reason is that whenever two edges that are incident to different faces cross, either the embedding has been changed (but this is forbidden by (H2)) or there must be some crossing between two edges that are incident to the same face. So instead of modeling (H4) for *all* pairs of non-incident edges, we model it only for pairs of non-incident edges that are incident to a common face. According to the above observation an embedding-preserving layout without any crossing of two edges incident to the same face cannot contain any crossing at all.

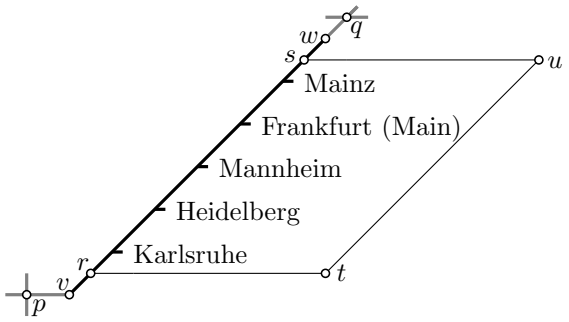
Even with this primary size reduction the models for most of our metro map examples were still too large to yield fast solutions. We observed that, on the one hand, only a small fraction of all possible spacing conflicts was relevant for the layout, that is, edge pairs for which (H4) had to be modeled explicitly. On the other hand, it is not clear how to determine these relevant edge pairs in advance. Fortunately, we could take advantage of the *callback* functionality of the MIP optimizer CPLEX as follows. In the initial MIP formulation we do not consider (H4) at all. Then, during the optimization process, we add constraints on demand, that is, as soon as the optimizer returns a new candidate solution, a callback routine is notified. This routine interrupts the optimizer and checks externally for violations of (H4) in the current layout. If there are pairs of edges that intersect, we add the respective edge spacing constraints for those pairs and reject the candidate solution that has now become infeasible. Then we continue the optimization. Our case study in Section 3.7 shows the impressive performance gain of this approach.

3.6 Labeling

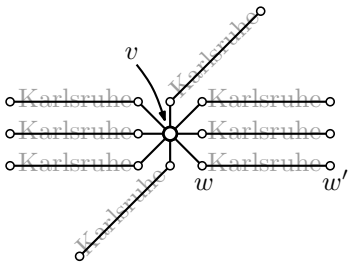
In practice, a metro map is of no interest to a passenger unless all stations are labeled by their respective names, see design rule (R6). The most fundamental requirement in a labeled metro map is that labels do not overlap other labels, vertices, or edges of the graph. Basically, there are two different ways of generating labeled metro maps: (a) using a two-phase approach that first generates an unlabeled layout and then, as a second step, places the labels within this layout as good as possible, or (b) using an integrated *graph labeling* approach that directly generates a labeled layout. Only the latter integrated approach assures that there is enough space to place all labels without overlap.

We follow the graph labeling approach and realize the hard constraint (H5) by enhancing the metro graph with labeling regions that are large enough to accommodate all labels that are assigned to them. For this enhanced graph, we set up the MIP model as described before. Its solution will be a crossing-free layout, which means in turn that all labeling regions will be empty and their labels can safely be placed inside.

Here, we assume that the degree-2 vertices have been collapsed as described in Sec-



(a) Vertex labels between interchanges p and q are modeled as a parallelogram-shaped region attached to edge vw.



(b) The vertex label of an interchange v (central vertex) is modeled as a two-link path (v, w, w') with eight possible shapes.

Figure 3.8: Label placement as a group of labels (a) or individually (b).

tion 3.5.1. For each path of length 3 between two interchange stations we model its labeling region as a parallelogram attached to the middle segment of the path. Figure 3.8a shows an example of labeling the middle edge vw of the path (p, v, w, q). The collapsed vertices will later be inserted along this middle segment and all their labels lie to the same side of the path. Usually, this is visually more pleasing than an arbitrary mix of labels on both sides. The side length of the parallelogram matches the length of its longest vertex label. Both to keep the number of reading directions small and to avoid unnecessary complexity in the model, we restrict labels to be placed horizontally or, if the corresponding edge itself is horizontal, diagonally in z_1-direction. Note that our model extends the ideas of Binucci et al. [BDLN05] who use a similar MIP model to label edges with fixed-size rectangles in an orthogonal graph drawing. In our case the parallelograms that contain the labels can be seen as additional metro lines. They differ from the other metro lines in that they can flip sides and in that their shape is fixed. So consider the situation in Figure 3.8a. We first subdivide the edge $e = vw$ by inserting two dummy vertices r and s and make sure that e cannot bend at r and s. This is achieved by the constraints

$$\operatorname{dir}(v, r) = \operatorname{dir}(r, s) = \operatorname{dir}(s, w). \tag{3.16}$$

We add two more vertices t and u and the edges rt, tu, and su. Edges rt and su are forced to be horizontal and to be of length l_{rt}, the length of the longest vertex label on e. For rt this is accomplished with the constraints

$$\begin{aligned} y(r) &= y(t) \\ x(r) - x(t) &\leq \rho(e)M + l_{rt} \\ x(r) - x(t) &\geq -\rho(e)M + l_{rt} \\ x(t) - x(r) &\leq (1 - \rho(e))M + l_{rt} \\ x(t) - x(r) &\geq -(1 - \rho(e))M + l_{rt}, \end{aligned} \tag{3.17}$$

where $\rho(e)$ is a binary variable that decides whether the labels are to the left ($\rho(e) = 0$) or to the right ($\rho(e) = 1$) of e. For su the constraints are analogous to (3.17) using the same binary variable $\rho(e)$. The third edge tu is forced to be parallel to rs by the constraint

$$\operatorname{dir}(t, u) = \operatorname{dir}(r, s) \tag{3.18}$$

so that the new edges indeed form a parallelogram attached to e. This parallelogram can still be placed on either side of e, modeled by the binary variable $\rho(e)$. For horizontal

graph	**Vienna** (5 lines)				**Sydney** (10 lines)				**London** (11 lines)			
	n	m	N	f	n	m	N	f	n	m	N	f
original	84	90	90	8	174	183	289	11	308	361	441	55
reduced	48	54	54	8	88	97	186	11	267	320	409	55
labeled	136	155	87	21	242	270	286	30	672	771	593	99

Table 3.2: Size of the real-world graphs in terms of vertices (n), edges (m), total edge size (N), and faces (f) for the *original* graph, the *reduced* graph after removing degree-2 vertices, and the *labeled* reduced graph.

edges with z_1-diagonal labels the constraints are analogous. Clearly, we do not enforce the circular order constraints (H2) for r and s. Moreover, the new edges rt, tu, and su are not taken into account in the objective function. Finally, because an edge can be drawn horizontally or diagonally, we need to do another case distinction in order to select either the set of constraints for horizontal or for diagonal labels. This is, as usual, achieved by means of a binary variable.

For labeling a single vertex v, for example, an interchange station, we create new vertices w and w' and introduce edges vw and ww', see Figure 3.8b. The first edge vw is of unit length and we allow vw to take any position in the circular order of the edges incident to v. The second edge ww' models the actual label; its length is equal to the label length. The direction of ww' depends on the direction of vw: ww' is horizontal if vw is non-vertical and ww' is z_1-diagonal if vw is vertical. Choosing one of the eight possible positions is modeled by eight binary decision variables.

3.7 Results and Evaluation

In this section we report the results of three case studies performed for the metro networks of Vienna, Sydney, and London. Sydney is a medium-size example that was used before by Hong et al. [HMdN06] and Stott and Rodgers [SR04] to evaluate their methods. Hence we are able to compare our results for the Sydney network to their layouts. The size of the graphs is given in Table 3.2 and ranges from 84 vertices and 8 faces (Vienna) to 308 vertices and 55 faces (London). The table further shows how the removal of degree-2 vertices described in Section 3.5.1 effectively reduces the number of vertices and edges. The last row gives the size of the reduced graphs with station labels added as described in Section 3.6.

The input graphs are given by a list of vertices with x- and y-coordinates and station names, and by a list of edges, each of which is associated to the metro lines to which it belongs. The input embedding assumes straight-line edges. Recall that all edge crossings that exist in the input layout are replaced by dummy vertices and are thus preserved in our output drawings.

The environment for computing our layouts was a Linux system based on an AMD Opteron 2218 CPU with 2.6 GHz and 8 GB RAM. Our implementation is a Java program that generates the MIP, solves it using the commercial optimizer Ilog CPLEX 11.1[2], and

[2] see `http://www.ilog.com/products/cplex`

then produces the layout from the coordinates in the MIP solution. We chose a time frame of 12 hours for computing the layouts. If optimality could not be shown within this time we report the best integer feasible solution and the remaining optimality gap. Note that CPLEX typically generates intermediate solutions quickly while most of the computation time is spent on finding minor improvements to the objective function. In practice it is worthwhile to examine these suboptimal solutions,too, since our objective function is only a humble mathematical attempt to capture the aesthetics of a schematic network layout. Hence suboptimal layouts may in fact be visually more pleasing in some instances.

3.7.1 Vienna

Our first case study, and a rather small example, is the metro network of *Vienna*, which is depicted geographically in Figure 3.9a. Despite its small size it is still a representative example of the large class of metro systems with only a few metro lines, see Ovenden [Ove03]. The individual lines connect two "opposite" terminus stations in the periphery of the city leading through the city center, where interchanges with the other lines are available. The official schematic layout by the transport company Wiener Linien is shown in Figure 3.9b. Note that this map includes additional surface train lines drawn as thin blue lines. These are not present in our input network. We thus restrict our comparison to the drawing of the five thick metro lines.

We show an unlabeled and a labeled layout produced by our method in Figure 3.10. The weights in the objective function were chosen as $(\lambda_{(S1)}, \lambda_{(S2)}, \lambda_{(S3)}) = (3, 3, 1)$. The unlabeled layout in Figure 3.10a was obtained by CPLEX within 3 seconds and the optimality of the solution with respect to the given weights for the objective function was proven within 22 seconds. It was not necessary to include any of the constraints (H4) in order to find a planar layout. The actual size of the MIP model that was solved is given in the bold column labeled *none* in Table 3.3. The first intermediate solutions for the labeled layout were returned after 74 seconds; the final labeled layout in Figure 3.10b, however, took CPLEX 10 hours and 8 minutes to compute. The optimality of this solution could not be proven and some manual improvements are obviously possible, for example, removing the bend of the orange line at Westbahnhof, the interchange with the brown line; the remaining optimality gap was still 25.4%. Here, it was necessary to add the constraints (H4) for 165 pairs of edges in order to find a planar layout, see the column labeled *callback* in Table 3.3 for the actual size of the model. Note that the callback method is not able to reduce the number of variables, but the number of constraints was reduced to less than ten percent of the original constraints (see column *faces*). Without using the callback method, no labeled layout of Vienna was found at all within our 12-hour time frame.

Now we compare the official layout (Figure 3.9b), the unlabeled layout (Figure 3.10a) and the labeled layout (Figure 3.10b) according to the seven design rules (R1)–(R7) given in Section 3.3.1.

(R1) Octilinearity All three maps use exclusively octilinear edges.

(R2) Topology All three maps preserve the input topology.

(R3) Line bends The number of bends is larger in the official layout (16 bends), than in our layouts (both 13 bends). All bend angles in all three layouts are 135° as requested by rule (R3). The official layout has four bends in interchanges, the unlabeled layout has three such bends, and the labeled layout two. Note, however, that the affected interchanges are degree-4 vertices in which only one of the two crossing lines has a

number of	unlabeled			labeled			
	all pairs	faces	**none**	all pairs	faces	**callback**	none
variables	11,584	6,584	**880**	94,948	29,908	**29,908**	1,572
constraints	45,582	24,957	**1,428**	388,564	120,274	**12,311**	3,388
edge pairs	1,338	713	**0**	11,672	3,542	**165**	0

Table 3.3: Size of MIP models for the Vienna metro map in terms of *variables*, *constraints*, and *edge pairs*. The columns represent the different models in which (H4) is in effect for *all pairs* of edges, for those incident to a common *face*, for those selected during the optimization by a *callback*, or for *none*. Columns corresponding to the shown examples are marked in bold.

bend while the other line goes straight. Still, in terms of bend minimization, our layout achieve slightly better results than the official layout.

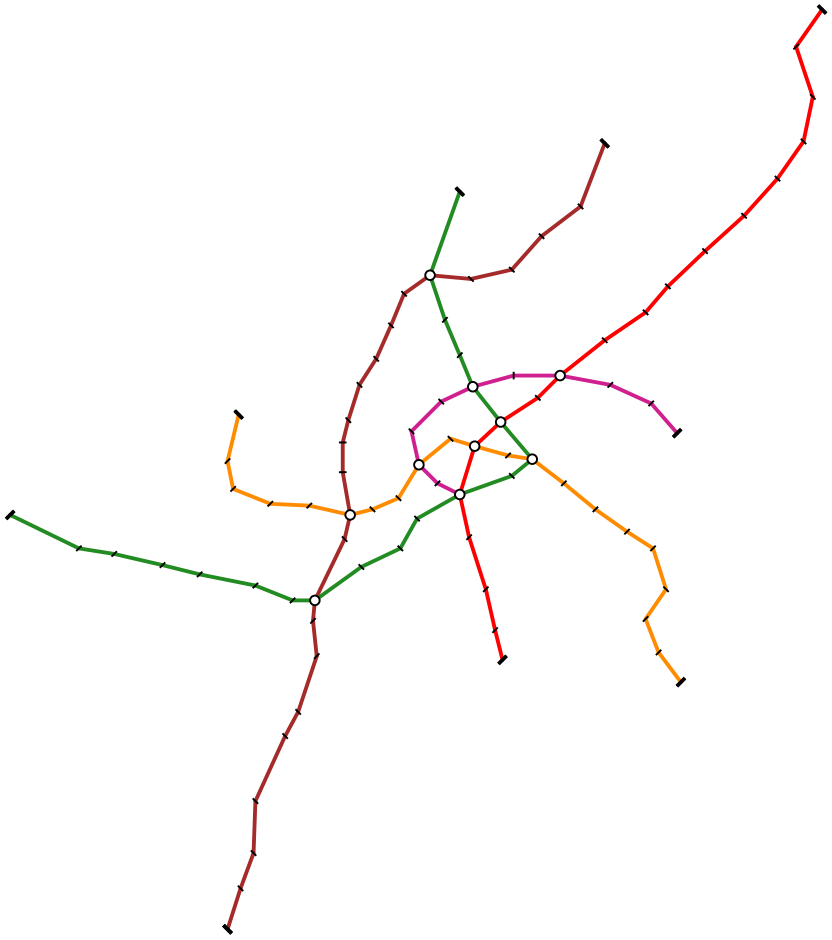
(R4) Relative position The relative position rule is judged according to the similarity with the geographic map (Figure 3.9a). The downside of the official map that it uses more bends than our layouts (rule (R3)) conversely means that it achieves a relatively high similarity to the geography in certain parts of the network (for example, the south-eastern end of the orange line U3 or the northern end of the brown line U6). The S-shaped orange line, however, is more realistically depicted in our unlabeled layout than in the official layout. A similar observation holds for the green line U4, which is drawn horizontally from its terminus Hütteldorf to the interchange Karlsplatz in the official layout. Both our layouts, on the other hand, show its south-west to north-east nature between the interchanges Längenfeldgasse and Landstraße more accurately. All in all, the official layout reflects some local features more accurately than our layouts, but our unlabeled layout better shows the general course of the lines. Our labeled layout has a similar appearance as the unlabeled layout, with the exception of the western end of the orange line, which is horizontally instead of diagonally to the north-west.

(R5) Edge lengths The edge lengths in the official and in our unlabeled layout are quite uniform. The labeled layout expands some of the edges up to seven times the unit length. This is due to the space requirements imposed to the MIP model by the label lengths.

(R6) Station labels Clearly, the unlabeled layout does not satisfy this rule. Both the official and our labeled layout use non-overlapping labels. By construction all labels between two interchanges are on the same side of their line in our layout. The official layout generally sticks to this rule as well, but occasionally swaps sides in order to obtain a more compact map.

(R7) Line colors In the Vienna network there are no parallel lines; since we are using the same colors as the official map there is no difference here.

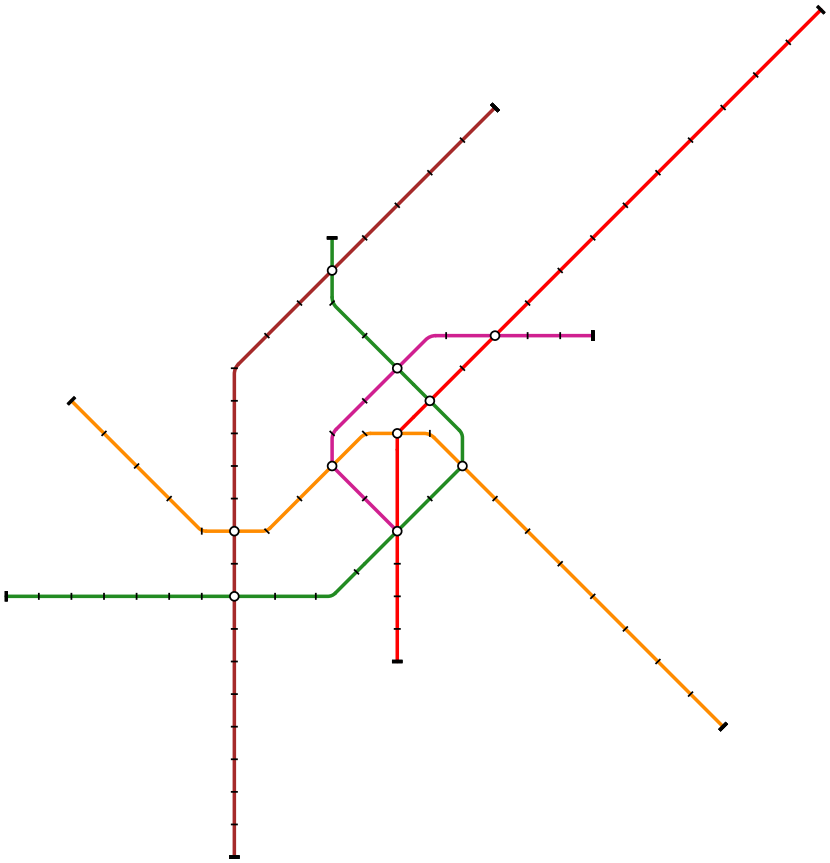
For a general evaluation of the three layouts it must be noted that the official map looks more cramped, but this is caused by the extra surface lines that are shown. As for the layout of the metro graph our unlabeled layout is a well-balanced compromise of rules (R3) and (R4). With only minor adjustments it could be used by a graphic designer as a basis



(a) Geographic input layout.

(b) Official layout (only thick lines belong to the metro system). Courtesy of Wiener Linien [Wie08].

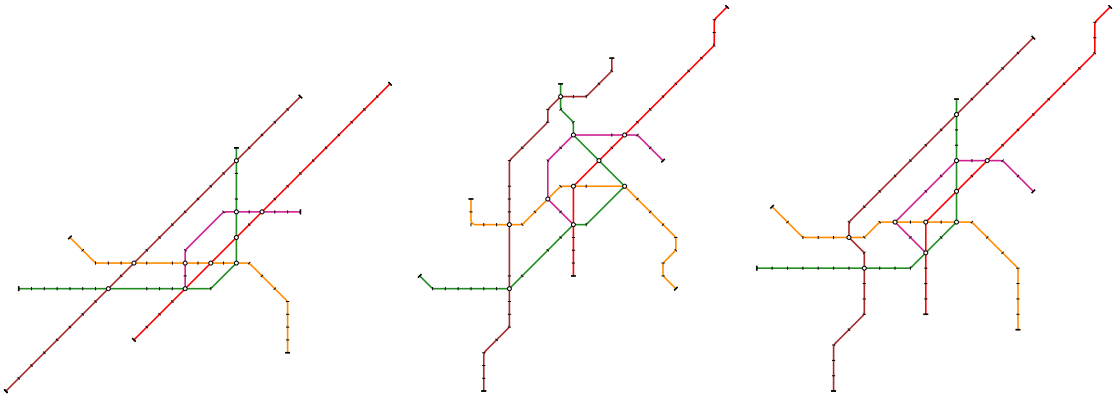
Figure 3.9: Layouts of the metro network of Vienna.



(a) Unlabeled layout.

(b) Labeled layout.

Figure 3.10: Layouts of the metro network of Vienna produced by our method.



(a) Weights $(10, 1, 1)$. (b) Weights $(1, 10, 1)$. (c) Weights $(1, 1, 10)$.

Figure 3.11: Layouts of the original metro network of Vienna with emphasis on bend minimization (a), preservation of relative positions (b), and length minimization (c) by assigning different weight vectors $(\lambda_{(S1)}, \lambda_{(S2)}, \lambda_{(S3)})$.

for a manually labeled map. Our automatically labeled layout shows the potential of our approach for producing valid labeled maps that satisfy rule (R6), albeit at the expense of losing to some extent the aesthetic quality of the unlabeled layout. Especially the uniform edge length rule (R5) is violated by some rather long edges, for example, on the brown line between Westbahnhof and Längenfeldgasse or the purple line between Karlsplatz and Volkstheater. Another reason for the slightly lower quality of our labeled layout is that in manually designed metro maps long names are often broken into two lines, which is currently not supported by our model. Obviously, breaking long lines makes labels easier to fit into a compact drawing that has more uniform edge lengths and a shape that is more similar to the unlabeled layout.

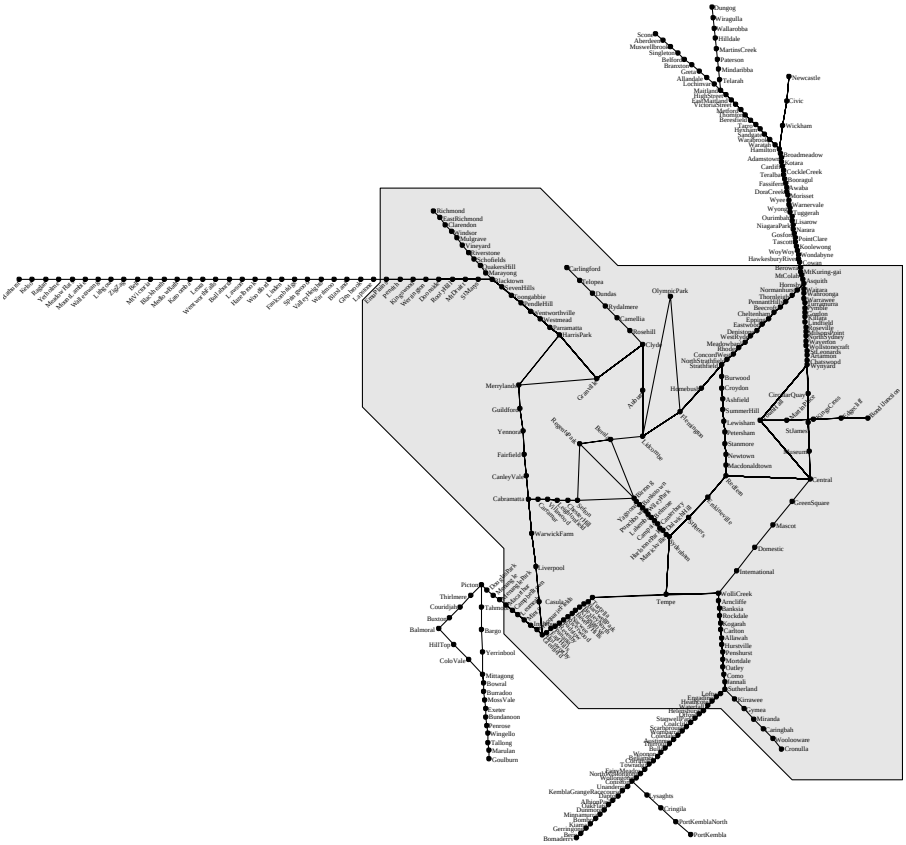
Before moving to the next case study, we illustrate the influence of the three soft constraints (S1)–(S3) on the network layout. Figure 3.11 shows three layouts, each of which exaggerates one of the soft constraints. Note that in order to better demonstrate the effects these layouts are created from the original metro graph, that is, none of the degree-2 vertices are removed. The first layout in Figure 3.11a optimizes line straightness. Indeed the red and brown lines have no bends. From the geographic orientations of the edges (see Figure 3.9a) it is clear that the bends in the remaining lines cannot be straightened given that our model restricts each edge to only three admissible directions (recall Section 3.4.2). In the next layout in Figure 3.11b the emphasis is on reflecting the original edge directions, which is also clearly visible. Of course, this results in an increase of the number of bends. The last layout emphasizes a small total edge length. It can be seen in Figure 3.11c that indeed only four edges in the center of the map have a length of two units and all others are of unit length. Some bends are introduced in order to compress the edges in the inner part of the network. It is obvious that none of these three extreme examples is a good layout. It requires a carefully balanced weight vector in order to obtain drawings that meet the quality requirements. In the end it is a matter of taste whether there should be a slight tendency towards bend minimization or towards preservation of the mental map.

3.7.2 Sydney

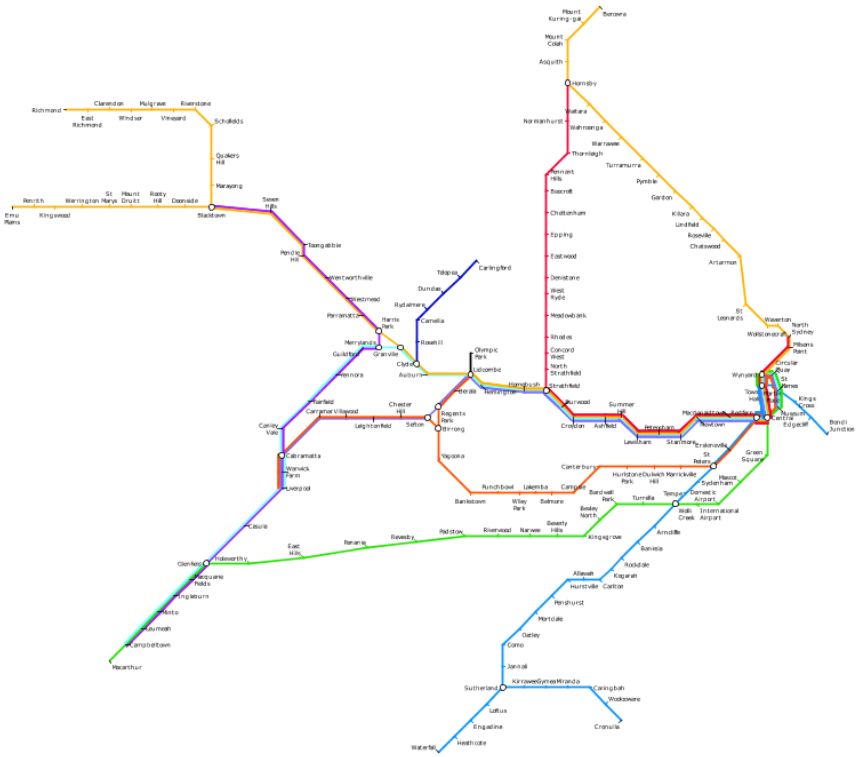
Our next case study is the CityRail System of Sydney that has already been introduced as an example in Section 3.3. In our discussion below we refer to the geographic and the official schematic layout of the network shown in Figure 3.3 of Section 3.3. The CityRail network is a medium-size public transport system (see Table 3.2) whose characteristics differ in parts from the previous example of Vienna. One new property is that there are quite a few parallel lines along central backbone paths of the network. Moreover, due to the geographic setting of Sydney on the coast, many lines lead from a peripheral terminus to a downtown terminus close to the sea.

Since Sydney has been used before as an example to demonstrate the layout techniques of Hong et al. [HMdN06] and of Stott and Rodgers [SR04] we can use it as a benchmark to compare their results with ours. Figure 3.12a shows the Sydney network laid out by the force-directed method of Hong et al. [HMdN06]. Note that they used a slightly larger network that includes additional intercity connections. The suburban part of the network, which is the basis of our comparison, is highlighted in gray. Unfortunately, no explicit results for the suburban network are available. Still, we may argue that the layout of the central part would look very similar to Figure 3.12a since the four additional branches in the periphery do not exert any significant repelling or attracting forces to the edges of the suburban part. The algorithm of Hong et al. is very fast and it took only 7.6 seconds to compute their layout on a 3GHz Pentium 4 machine with 1GB of RAM. Figure 3.12b shows the most refined layout produced by the methods of Stott and Rodgers [SR04, SR05, Sto08]. In this example they did not apply any preprocessing to collapse degree-2 vertices. Stott [Sto08] does not report the actual computation time for this layout; he mentions, however, that for a network with more than 100 vertices (Sydney: 174 vertices) each iteration takes between one and four hours on a 1.8GHz machine and that there are several iterations necessary before the layout converges. The first version of their algorithm, which produced unlabeled maps only, took about 28 minutes for an unlabeled map of the Sydney network [SR04].

We show an unlabeled and a labeled layout of the Sydney CityRail network produced by our method in Figure 3.13. For the unlabeled layout (Figure 3.13a), the weights were chosen as $(\lambda_{(S1)}, \lambda_{(S2)}, \lambda_{(S3)}) = (3, 2, 1)$, which slightly emphasizes minimizing bends before preserving relative positions. This layout was obtained in 23 minutes and 22 seconds. Optimality of this solution could not be proven within our time limit. The remaining optimality gap was still 16.4% after 12 hours. The callback method needed to add the constraints (H4) for 3 pairs of edges, see the bold column in Table 3.4. Note that for the unlabeled layout we did not consider all possible pairs of edges that share a common face as candidates for (H4) but only those that involve at least one *pendant* edge, that is, an edge on the path between a degree-1 vertex and the first interchange. This is based on the observation that in unlabeled layouts the pendant edges tend to be the ones that cause crossings (in this case the dark blue line in the center of the layout). This reduced the number of variables from otherwise 20,554 to only 4,834, see Table 3.4. For the labeled layout we changed the weight in the objective function to $(\lambda_{(S1)}, \lambda_{(S2)}, \lambda_{(S3)}) = (3, 3, 1)$. The final labeled layout in Figure 3.13b was computed by CPLEX in 10 hours and 31 minutes, while the first suboptimal solutions were found after 3 minutes. As before, optimality of this layout could not be proven and an optimality gap of 15.5% remained after 12 hours. The constraints (H4) were added during the optimization for 123 pairs of edges by the callback mechanism, see Table 3.4. This corresponds to a reduction of the number of constraints to less than six percent with respect to the column *faces*.

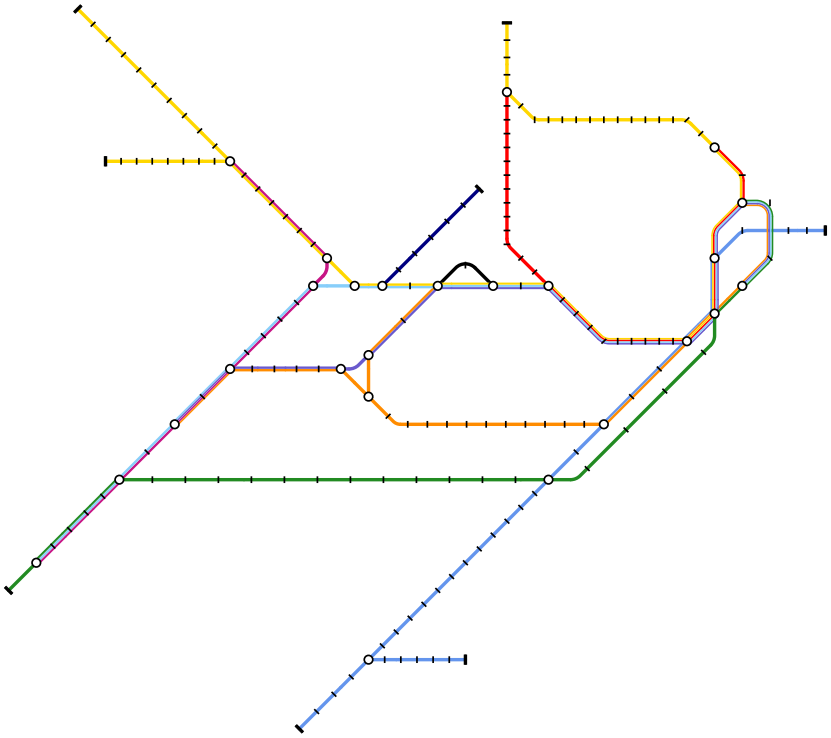


(a) Layout by Hong et al. [HMdN06]. The gray area highlights the suburban part.

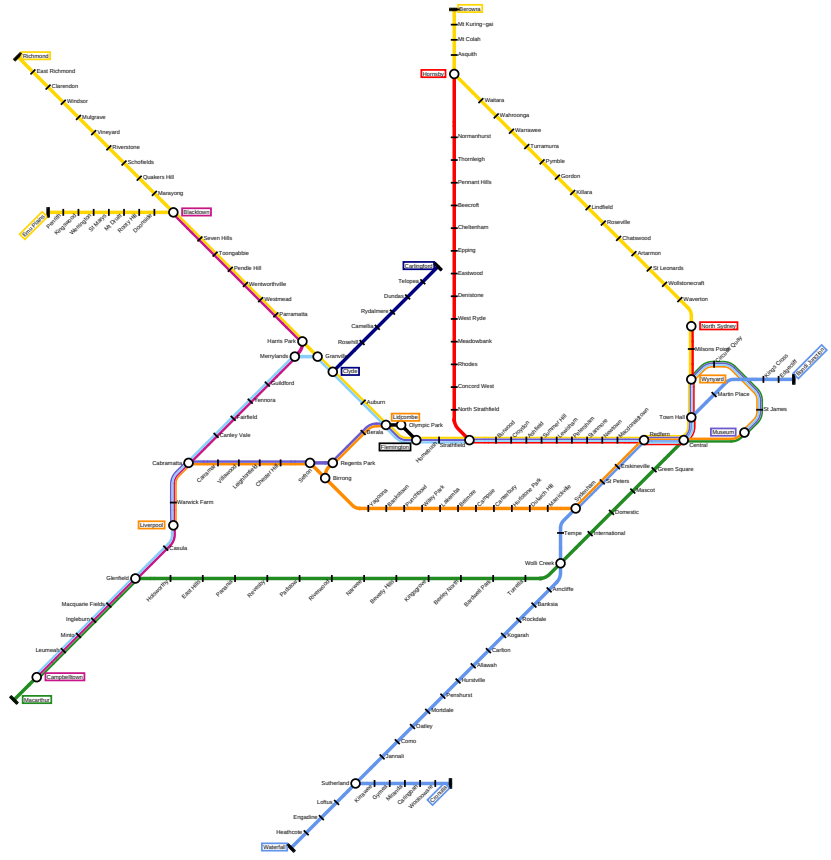


(b) Layout by Stott and Rodgers [SR04, SR05] (figure taken from Stott [Sto08]).

Figure 3.12: Layouts of the Sydney CityRail network produced by previous methods.



(a) Unlabeled layout.



(b) Labeled layout.

Figure 3.13: Layouts of the Sydney CityRail network produced by our method.

number of	unlabeled			
	all pairs	faces	**callback**	none
variables	37,802	20,554	**4,834**	1,642
constraints	152,194	81,046	**3,529**	3,034
edge pairs	4,520	2,364	**3**	0
number of	labeled			
	all pairs	faces	**callback**	none
variables	290,137	92,681	**92,681**	2,969
constraints	1,191,406	376,900	**21,988**	6,838
edge pairs	35,896	11,214	**123**	0

Table 3.4: Size of MIP models for the Sydney metro map in terms of *variables*, *constraints*, and *edge pairs*. The columns represent the different models in which (H4) is in effect for *all pairs* of edges, for those incident to a common *face*, for those selected during the optimization by a *callback*, or for *none*. Columns corresponding to the shown examples are marked in bold.

Next we present a detailed evaluation of the official layout (Figure 3.3b), the layout by Hong et al. [HMdN06] (Figure 3.12a), the layout by Stott and Rodgers [SR04, SR05] (Figure 3.12b), and our unlabeled and labeled layouts (Figures 3.13a and 3.13b) according to the seven design rules (R1)–(R7).

(R1) Octilinearity

- The official layout uses octilinear edges only.
- The layout of Hong et al. mostly uses edge directions that are close to but not quite octilinear. Some edges clearly lie in between two octilinear directions. This effect seems to be due to the fact that the forces that determine the layout are the sum of many conflicting terms, only one of which drags edges into an octilinear direction.
- The layout of Stott and Rodgers draws most edges octilinearly; few edges are not octilinear.
- By construction all edges in our layouts are octilinear.

(R2) Topology

- All layouts preserve the input topology. [3]

(R3) Line bends

- The official layout successfully avoids line bends whenever possible; only two bends on the north-western end of the yellow line seem unnecessary. Most bends have turning angles of 135°, only few bends make 90° turns (for example, in the

[3] Although, accidentally, Figure 3.12a seems to contain two incorrect edges.

loop on the right), and one angle is only 45° (orange line in the center of the layout).

- The layout by Hong et al. does not have any bends between two interchanges due to the removal of all degree-2 vertices. Unfortunately, their layout does not show the metro lines explicitly, which makes rule (R3) hard to evaluate. Although Hong et al. [HMdN06] list avoiding line bends as an aesthetic criterion, their method does not explicitly take the metro lines in account and so lines do have several (often acute) bends in interchange stations.
- The method of Stott and Rodgers takes line bends into account and partially succeeds in avoiding them; however, the algorithm is susceptible to local minima, in which shifting a single vertex is not sufficient to remove some obviously unnecessary line bends. Most bends form a 135° angle as desired.
- Our layouts, in particular the labeled layout, have low numbers of line bends that are comparable to the official map. The loop of the red line and the yellow line in the north-east has a larger number of bends in the unlabeled map than in the labeled one. With only few exceptions the bends form 135° angles.

(R4) Relative position

- The official layout preserves the general sense of the geographic map fairly well. Only the orange line in the center of the layout is straightened rather strongly.
- The layout of Hong et al. strongly distorts the mental map of Sydney. It must be noted, though, that optimizing (R4) is not an objective of their method.
- Stott and Rodgers implemented a geographic relationship rule. Their layout indeed preserves the geographic layout well, showing also minor changes of direction.
- Our layouts are similar in shape to the official layout. The unlabeled layout has some noticeable distortions the north-eastern part, for example, the yellow line is drawn horizontally while it runs diagonally in the geographic map. The labeled layout does not have these distortions and thus better satisfies rule (R4). The course of the orange line in the center, which has been distorted in the official map, is more accurately reflecting the geography in both our layouts.

(R5) Edge lengths

- The edge lengths in the official layout are quite uniform. Only the edges of the vertical blue line in the south-east are very short. No overly long edges are found.
- Edge lengths in the layout of Hong et al. do not have a very uniform appearance. While stations on the peripheral ends are densely packed such that individual edges are even hard to recognize, some edges in the central part are very long compared to the rest of the layout. Especially edges incident to triangular faces are too long and give the layout an unbalanced appearance. Note that uniform edge lengths were not mentioned as an objective of their algorithm.
- Edge lengths in the layout of Stott and Rodgers are quite uniform. Only the edges forming the prominent loop in the east of the network are too short to be well recognizable.

- Edge lengths in our layouts are quite uniform. Only the edges of the loop in the east appear rather long in the unlabeled layout; the labeled layout seems slightly more balanced in terms of edge lengths.

(R6) Station labels

- The official map contains non-overlapping horizontal and diagonal labels. For the majority of paths between two interchanges, all labels lie on the same side of the path.
- With a few exceptions the horizontal and diagonal labels in the layout of Hong et al. are non-overlapping; some labels, however, do occlude edges of the graph or even other labels. Labels are mostly placed on the same side of a line, with some exceptions where they alternate between both sides. Note that in the algorithm of Hong et al. the graph layout is determined in a first step independently of the labeling; labels for the fixed graph layout are added in an independent second step.
- Stott and Rodgers use non-overlapping horizontal labels. In the eastern loop, however, labels do occlude edges. Labels tend to be placed on the same side of a line with the exception of horizontal lines, where an alternating placement above and below the line was necessary. Some ambiguous labels exist.
- Clearly, our unlabeled layout fails to satisfy (R6). The labeled layout uses non-overlapping labels by construction. There is a horizontal and a diagonal label in the upper part of the eastern loop (Milsons Point and Circular Quay) that are very close to each other; increasing the label length by a safety offset would avoid this. Again by construction, all labels between two interchanges are placed on the same side of the line. A few interchange stations are slightly ambiguously labeled.

(R7) Line colors

- The official map uses distinctively colored lines and strongly increases edge widths where multiple parallel lines (up to six) are present.
- The layout of Hong et al. fails to use colors. Only the underlying network is drawn and individual lines cannot be recognized. This is merely an rendering problem.
- Stott and Rodgers use distinct colors and widen edges where multiple parallel lines are present. For edges with six parallel lines, the individual lines are hardly visible.
- Our layouts use distinct colors and edges are widened to accommodate parallel lines. For edges with six parallel lines, individual lines become hard to recognize, similarly to the layout of Stott and Rodgers.

As to be expected, the manually designed official layout turns out to balance all seven design rules very well and there is only very little room for obvious improvements. An interesting feature of the official map is the inclusion of the coastline to support the mental map of the users.

The method of Hong et al. [HMdN06] has the advantage that layouts can be computed very fast (7.6 seconds in the case of Sydney); the visual quality, however, is far from complying with our design rules. Note that the quality criteria considered by Hong et al. were *line straightness* (similar to rule (R3)), *no edge crossings* (implicit in rule (R2)), *non-overlapping labels* (rule (R6)), and *octilinearity* (rule (R1)). While there are no edge crossings created and the labeling has (almost) no overlaps, there is still room for improvements in terms of octilinearity and line straightness. It is an interesting problem to model the remaining design rules in a fast force-directed method as the one of Hong et al.

The layout by Stott and Rodgers [SR04, SR05] achieves a higher quality than the one of Hong et al. and it is more similar to the official layout. The map fulfills rule (R4) quite well, but does so at the expense of a large number of bends (rule (R3)). Another disadvantage is that not all edges are octilinear and that the prominent loop in the east of Sydney is not enlarged enough to be clearly visible. Computation times for their algorithm are not explicitly stated, but it seems to require several hours for a layout as the one in Figure 3.12b.

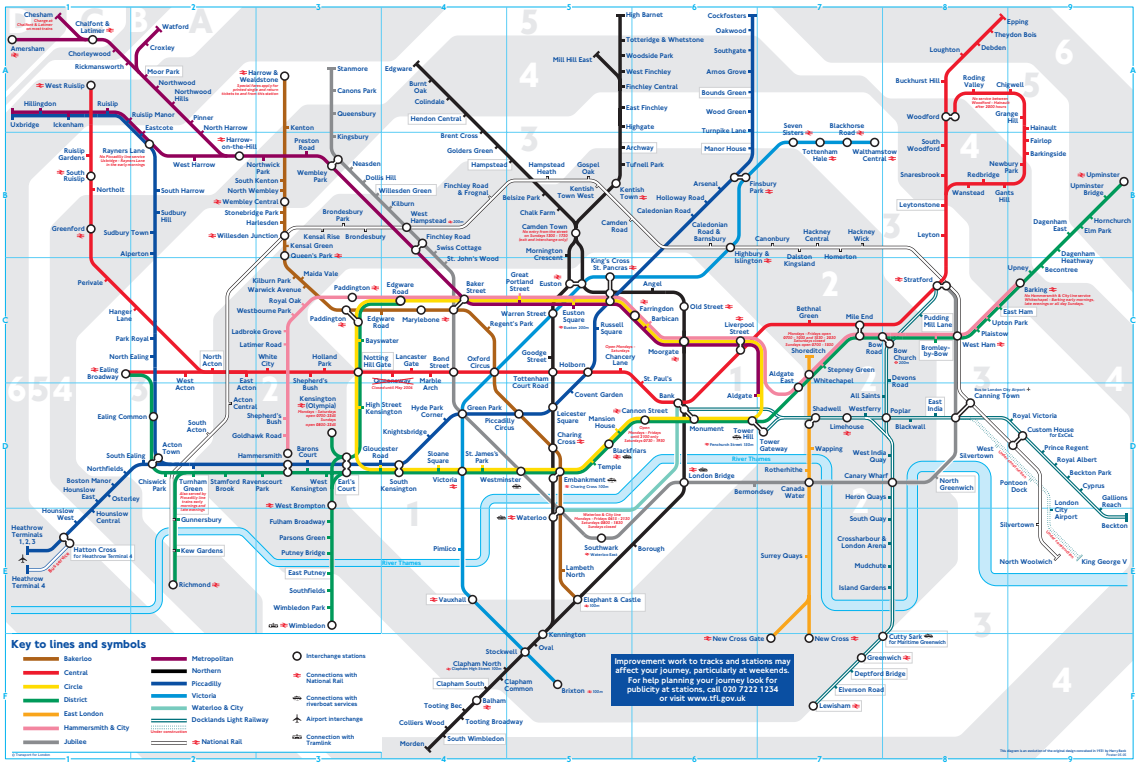
Finally, the evaluation of the seven design rules shows that our method is indeed able to produce labeled metro maps whose visual quality is comparable to manually designed maps. The design rules that are modeled as hard constraints are satisfied by construction and even the design rules (R3), (R4), and (R5) that are modeled as soft constraints are well balanced by the solution produced in the global optimization of our MIP. The main deficiency that remains is the handling of edges with more than four parallel lines. Such edges require significantly more space than the line segment as which they are currently modeled. The computation time for our labeled map was about 10.5 hours and thus several orders of magnitude higher than the running time of Hong et al. [HMdN06]; our running times seem comparable to those reported by Stott [Sto08] though. The computation time for the unlabeled map was significantly shorter, although in the example of Sydney (unlike Vienna in the previous section) some parts of the network even seem to satisfy the design rules better in the labeled layout than in the unlabeled layout.

3.7.3 London

Our last example is the famous “tube map” of the London Underground network, for which Henry Beck has designed the first schematic metro map in 1933 [Gar94]. Since the times of Beck the network as well as the map have undergone many changes, see Roberts' book on the history of the tube map [Rob05]. With its 308 vertices and 55 faces (see Table 3.2) London is one of the world's largest and most complex metro systems. The network is depicted geographically in Figure 3.14a. The metro lines mostly connect two peripheral stations leading through the densely connected city center, which is bounded by the yellow Circle line. A striking feature of the official tube map shown in Figure 3.14b is the flask-shaped layout of this Circle line in the center.

Figure 3.15 shows an unlabeled layout of the London Underground network. The weights in the objective function were chosen as $(\lambda_{(S1)}, \lambda_{(S2)}, \lambda_{(S3)}) = (3, 2, 1)$, thus emphasizing bend minimization stronger than preservation of the input geometry. It took 10 hours and 24 minutes to compute this layout and the remaining optimality gap after 12 hours was still 21.3%. It was necessary to add the constraints (H4) for 15 pairs of edges using the callback method. The size of the corresponding MIP model is highlighted in bold in Table 3.5. As in the previous example of the unlabeled Sydney layout, we considered only pairs with at least one pendant edge for that callback constraints, which reduced the number of variables

(a) Geographic input layout.



(b) Official layout by Transport for London [Tra08].

Figure 3.14: Layouts of the London Underground network.

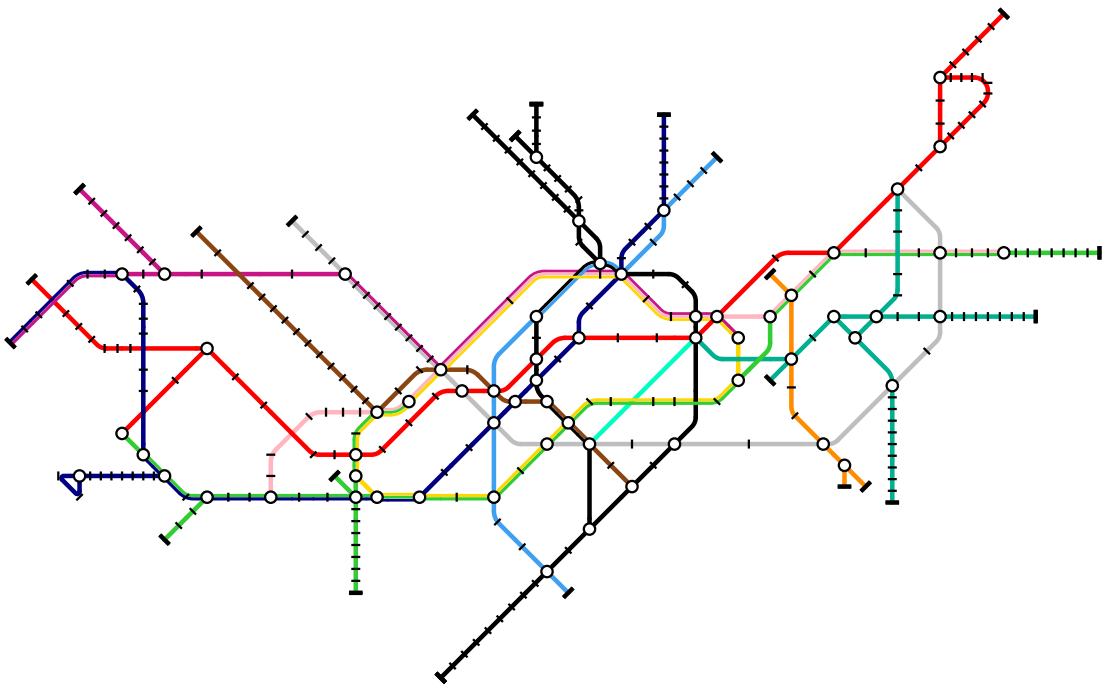


Figure 3.15: Unlabeled layout of the London Underground network produced by our method.

number of	unlabeled				labeled		
	all pairs	faces	**callback**	none	all pairs	faces	none
variables	408,789	90,445	**15,821**	5,173	2,372,247	369,775	8,455
constraints	1,673,608	360,439	**10,503**	8,692	9,769,241	1,509,044	18,599
edge pairs	50,452	10,659	**15**	0	295,474	45,165	0

Table 3.5: Size of MIP models for the London metro map in terms of *variables*, *constraints*, and *edge pairs*. The columns represent the different models in which (H4) is in effect for *all pairs* of edges, for those incident to a common *face*, for those selected during the optimization by a *callback*, or for *none*. The column corresponding to the shown example is marked in bold.

from 90,445 to 15,821. Unfortunately we were not able to produce a *labeled* layout for the London Underground network. With still almost 370,000 variables and more than 45,000 possibly relevant edge pairs, the model is simply too large and too complex to solve for the current version of CPLEX. Placing station labels for the high-degree interchange vertices within the many small faces in the center of the map is a very challenging task, for which our rather simple labeling model has not proven itself suitable so far.

We now compare our unlabeled layout (Figure 3.15) with the official layout (Figure 3.14b) in terms of the seven design rules.

(R1) Octilinearity Both layouts use exclusively octilinear edges.

(R2) Topology By construction our layout maintains the input topology. Interestingly, the official layout is altering the topology by flipping the brown Bakerloo line between

the stations Paddington and Baker Street from outside the Circle line into the interior of the Circle line.

(R3) Line bends In terms of line bends both layouts have strengths and weaknesses. On the one hand, the official map routes, for example, the red Central line almost horizontally through the center of the map[4], while in our layout this line is drawn as a sequences of horizontal and diagonal segments from the lower left to the upper right—a bit like a staircase. On the other hand, the gray Jubilee line or the eastern part of the dark blue Piccadilly line use less bends in our layout than in the official one. All in all, the official layout still seems to use less bends, especially in the dense downtown area. Bends in interchanges are clearly avoided in the official map; such bends are rather realized immediately in front of the stations, see, for example, Green Park in cell D4 of the official map. While this is also the case in many of the interchanges in our layout, some lines do bend in interchanges.

(R4) Relative position The official map is quite successful in preserving relative positions, although there is also some distortion present. Lines leading into the periphery are bent inwards in order to keep the bounding box of the map small. For example, the west end of the red Central line is bent northwards although it extends westwards in reality. Similarly, the green District line in the east is placed diagonally instead of horizontally as would be the obvious choice from its geographic course. For these peripheral parts of the network, our layout is more accurate. In the central part, however, some lines in our layout are oversimplified (for example, the dark blue Piccadilly line) and others have bends that are not present in their geographic course (for example, the red Central line).

(R5) Edge lengths In both maps edge lengths are uniform if possible, for example, along peripheral parts of the lines; in some situations in both maps, edges are drawn significantly longer than the unit length in order to avoid additional bends. In spite of their uniformity, the edge lengths of pendant edges in our layout seem rather short in comparison to edge lengths in the central part of the map.

(R6) Station labels Our unlabeled map fails to show station labels. The official map is fully labeled without overlaps. A characteristic of the London map is that only horizontal labels are used, that is, for horizontal lines the labels are alternately placed above and below the line. On non-horizontal lines the labels between two interchanges are mostly placed on the same side of the line.

(R7) Line colors The same distinct line colors are used in both maps. Parallel lines are drawn as parallel copies of the respective lines.

To summarize the comparison, the official map is clearly ahead of our unlabeled layout with respect to the seven design rules. Moreover, its general appearance as a whole is very balanced and clear. This is no wonder, given that its design has evolved over more than 75 years since Beck's first drafts. The tube map is a piece of art that has become an icon of London itself and the attempt to replace this sophisticated layout by an automatically generated one is very keen if not destined to fail. Nonetheless, our unlabeled layout not only shows that automatically producing a schematic map of very large transport systems is

[4]Henry Beck devised the Central line as the horizontal basis of his diagram and placed the remaining lines around it [Gar94]. Later designs stuck to this decision.

possible, but it also shows that a high overall visual quality can be achieved. In many places the rules modeled as soft constraints are satisfied well, and there is definitely potential in our method to assist graphic designers in their layout choices by producing draft layouts for different weight vectors. Computing labeled layouts for such complex metro networks using our model remains an open problem.

3.8 Concluding Remarks

In this chapter we have studied the combined metro map layout and labeling problem. Initially, we have identified seven basic design rules to which the majority of real-world metro maps adhere. These rules were then split into hard and soft constraints that either have to be satisfied or that need to be optimized. In the main part we described how the hard and soft constraints can be translated into the linear constraints and the objective function of a mixed-integer program that can subsequently be solved with general-purpose optimizers like CPLEX. In three detailed case studies we evaluated the results of our approach in comparison to the official, manually designed layouts that are provided by the transport companies as well as to layouts generated by two previously published methods. It turned out that our method is indeed able to generate labeled metro maps for small and medium-size metro networks that are of high visual quality and that can compete with the official maps—given some finishing by a graphic designer. This also indicates that the design rules that we identified are actually also applied by professional graphic designers. For large and complex networks (such as the London Underground network), we were only able to demonstrate that good *unlabeled* layouts can be generated; in spite of the size-reduction techniques that were applied, the MIP model is still too large to be efficiently solvable for a *labeled* version of the network. Ideas to further reduce the size of the model are necessary. For example, one could consider partially labeled maps that model labels only for stations that are known to be difficult to label.

In terms of practical applicability we note that our method is unable to produce good labeled maps instantaneously; the layouts presented in Section 3.7 were mostly generated within 10 to 12 hours, but solution times are generally hard to predict. Still, designing a new high-quality schematic map is usually a process in which running times of several hours are acceptable. Moreover, the first intermediate (but suboptimal) results are often quickly generated and the final layouts differ only marginally from some of the earlier layouts. If our method is seen as a tool to assist graphic designers, such suboptimal layouts often may just as well serve as drafts. Recall that the objective function is just an attempt to model the aesthetic quality of a layout in mathematical terms. Hence slightly better solutions in terms of the objective function are not necessarily more pleasing for a human than some close-by solutions.

Open problems. There is a number of open problems that remain. First of all the treatment of edges that are used by a large number of parallel lines is not satisfactory in our model. Instead of treating an edge as a line segment, such “thick” edges should be modeled as rectangles that actually consume space in the drawing. Analogously, stations on such thick edges must also be modeled as disks or polygons rather than as points. We refer to the MIP formulation of Binucci et al. [BDLN05], where rectangular vertices are modeled for orthogonal graph drawing. As an example, Figure 3.16 shows a clipping of the transport network of Frankfurt am Main which has a large number of parallel lines in the

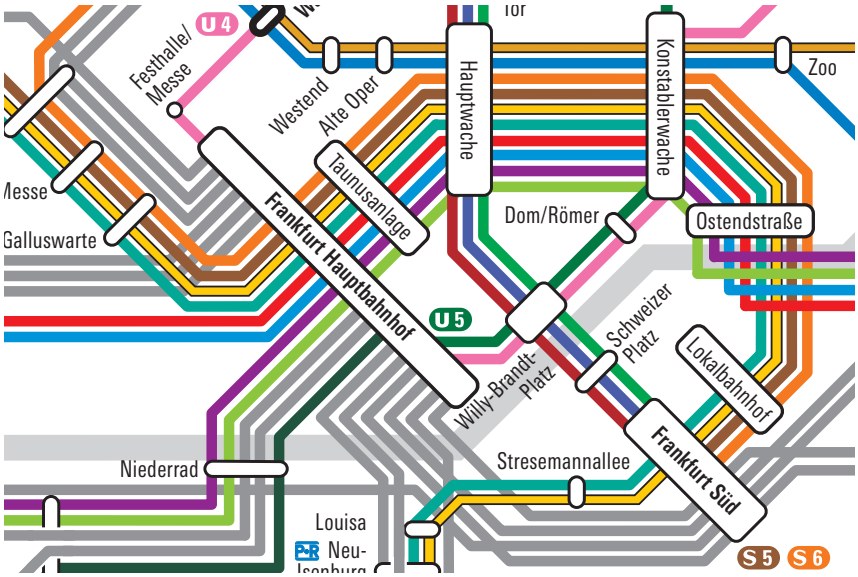


Figure 3.16: Clipping of the public transport map of Frankfurt am Main.

city center. The related problem of placing parallel lines with a minimum number of line crossings along the edges of the underlying graph layout is covered in Chapter 4.

Furthermore, graphic designers certainly apply more than our seven basic design rules when designing a schematic map. One obviously important criterion is the display of symmetries in the network (for example, the flask-shaped Circle line in London), or the inclusion of landmarks such as rivers, lakes, or coastlines. Such rules depend on domain knowledge of the network at hand and the artistic skills of the designer and seem very challenging to incorporate into a general-purpose mathematical model. Tufte [Tuf01] even says: "The principles [...] are not logically or mathematically certain; and it is better to violate any principle than to place graceless and inelegant marks on paper."

After all, it is upon the map users to decide what is a good and easy-to-use metro map. Some initial studies that compare geographic and schematic maps have confirmed the superiority of schematic maps over geographic maps for network navigation tasks [Bar80, Sto08]. Still, these findings are rather general; detailed and empirically confirmed guidelines for design criteria that make schematic maps easy-to-read and useful are missing. Hence a detailed user study that aims at ordering several design rules by their importance would be very helpful both for graphic designers and for devising suitable layout algorithms.

Possible extensions of our model include user interaction in a semi-automatic layout process. For instance, the user may specify a certain desired direction for some lines or edges, or a certain label position for some of the vertices. Such additional constraints can easily be included in our formulation. Area constraints that specify, for example, a maximum height of the layout, can also be included. Instead of minimizing the total edge length we can, alternatively, minimize one dimension of the map. This is useful if the map has to fit a certain area, for example, if it is placed above doors inside trains.

Chapter 4

Metro Maps: Line Crossings

Producing a schematic metro map is a multi-step process that poses algorithmic challenges at several levels. The previous chapter has focused on the combined layout and labeling problem, in which the underlying railway or road network is schematized such that all metro stations or bus stops can be labeled without overlap. Here, we consider a secondary problem that arises once the layout of the underlying network is fixed. In many metro networks, and especially in bus networks, it is common that edges, that is, physical tracks or roads, are shared by multiple transportation lines. The most common solution to visualize this situation in the map is to draw each transportation line in a distinct color. All lines that use an edge in the network are then drawn as a bundle along that edge in the underlying layout.

As an immediate consequence there are situations in which two lines in the—otherwise plane—network cross. Some line crossings are mandatory, induced by the network topology, others depend on the order of the lines and can be avoided by choosing a good line order. In the first part of this chapter we introduce the line crossing problem in metro maps and give a quadratic-time algorithm for determining an optimal line order along a single edge of the network. No results are known if two or more edges are considered. In the second part we study a variant in which all lines must be placed outermost in their respective terminus stations. This problem has been proven NP-hard by Bekos et al. [BKPS08]. We give an efficient algorithm to determine an optimal layout in general plane graphs if the input specifies where to place each line in its terminus, that is, on which side of the edge it is placed outermost. The chapter is based partly on joint work with Marc Benkert, Takeaki Uno, and Alexander Wolff [BNUW07] and partly on yet unpublished results with Joachim Gudmundsson, Damian Merrick, and Thomas Wolle.

4.1 Introduction

A public transportation network is, informally speaking, a network consisting of stations (or stops) that are served by a set of transportation lines, for example, by metro, bus, or tram lines. There is an edge between two stations in the network if they are adjacent in the sequence of stops of some transportation line. More formally, we represent the transportation network as a *metro graph* $(G, \mathcal{L})$, where $G = (V, E)$ is a planar embedded (or *plane*) graph of bounded degree (usually ≤ 8) and $\mathcal{L}$ is a set of paths in G. The vertices of G are the stations and bus stops in the network and the edges represent direct connections between their endpoints. We denote G as the *underlying network* since it

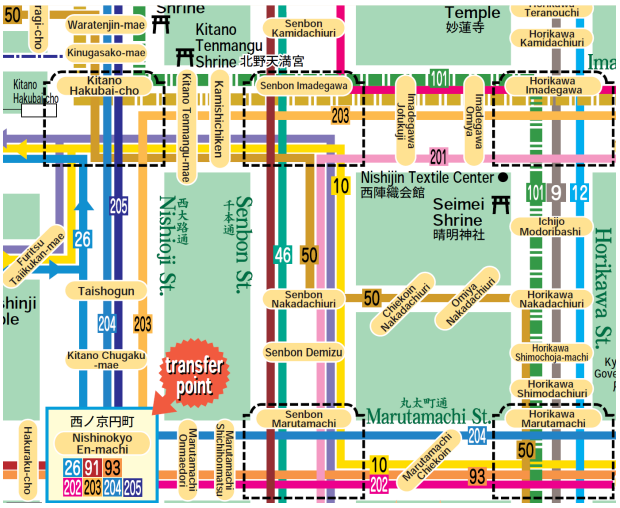


Figure 4.1: Kyoto bus map.

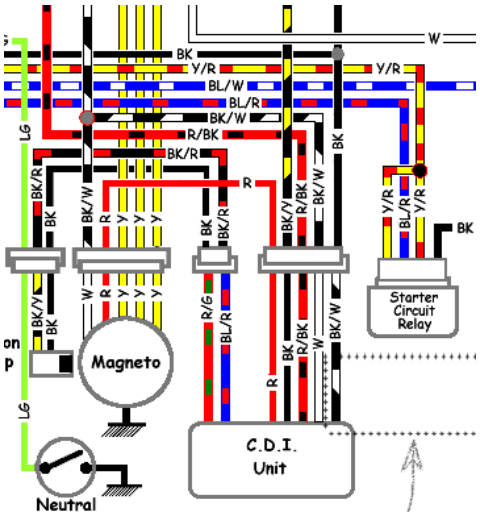


Figure 4.2: Wiring diagram of the Kawasaki KLR650 motorcycle.

corresponds to the underlying transportation infrastructure (railway tracks and roads) of the public transport system. The set $\mathcal{L}$ represents the transportation lines that serve the network. Since each edge in G is by definition served by at least one transportation line, $\mathcal{L}$ covers G and we denote $\mathcal{L}$ as the *line cover* of G.

A feature of many transportation networks is that parts of its underlying infrastructure are shared by multiple transportation lines, see Ovenden's collection of metro maps [Ove03]. Bus networks, even more, have this property, in particular in the surroundings of central hub stations, where many bus lines gather using only a small number of roads. Figure 4.1 shows a clipping of the bus network of Kyoto, Japan. In this figure the edges of the underlying network are drawn as white boxes in the background. The individual bus lines are drawn as distinctly colored parallel lines within the area occupied by the underlying edges. Vertices are usually drawn as disks or rectangles that are at least as wide as the incoming edge bundles. Note that drawing all transportation lines that use the same physical route as individual, distinctly colored parallel lines has already been introduced by Henry Beck, the designer of the first schematic map of the London Underground [Gar94].

The layout problem in this context is to determine an order of the incoming and outgoing lines of each vertex such that the resulting map is as easy to read as possible. A natural criterion by which to judge the quality of a layout is the number of line crossings, that is, crossings of transportation lines along shared subpaths, induced by the line orders in each vertex. Each line crossing is a visual obstacle that disturbs the viewer's eye following the course of a transportation line. Avoiding edge crossings in general has been empirically shown as one of the most important criteria for graph readability [PCJ97, Pur97]. So our goal is to find a set of line orders for all vertices that minimizes the number of line crossings. For example, in Figure 4.1 the number of line crossings can be reduced by four if line 50 (brown) in the center of the figure stays right of line 10 (yellow) and the unnumbered blue line just next to line 10.

Note that it is obviously possible to hide all line crossings below the areas occupied by the vertices. This means, however, that incoming and outgoing line orders of a vertex differ. Thus at each affected vertex the viewer first has to locate the correctly colored line

before following it to the next vertex. In contrast, if the incoming and outgoing line orders are identical, each line enters and leaves the vertex where expected. This simplifies reading the map; therefore we insist that no line crossings are hidden below vertices. Clearly, this does not hold for crossings between two lines that intersect in a single vertex, for example, the vertical line 12 (blue) and the horizontal line 10 (yellow) in Figure 4.1 must intersect below the area of the common vertex.

Related problems. Transportation networks are not the only applications in which a set of paths follows the edges of an underlying graph. Another example are wiring diagrams as the one depicted in Figure 4.2 or very-large-scale integration (VLSI) chip layouts. A wiring diagram is a visual aid to assist a mechanic in troubleshooting and fixing circuits that connect different electrical components, for example, in a motorcycle as in Figure 4.2. While some vertices in such a layout might require a fixed wire order as given by the connectors, other vertices can have variable wire orders. So similar to transportation networks, a legible wiring diagram should have as few wire crossings as possible.

In VLSI design the *via minimization problem* is also related to line crossing minimization. Printed circuit boards often consist of several stacked layers, each of which contains a crossing-free layout of wires. Since the board layout comprising the union of all layers is rarely planar, wire crossings are realized by using separate layers for the affected wires. In order to move to a different layer, small holes called *vias* are drilled into the board through which the wires can change their layer. Vias can cause electrical instabilities and hence via minimization is a common optimization criterion in circuit board design [Len90]. There are efficient algorithms for via minimization if only two layers are used but it becomes NP-hard for more than two layers [Len90, Chapter 9]. Conductive traces on such boards often consist of multiple wires connecting the different components on the board, which is a similar setting as traffic routes used by multiple transportation lines connecting large interchanges. By using only two vias, however, a wire can cross a whole bundle of other wires, whereas in line crossing minimization we consider each line crossing individually.

Model. In this chapter we assume that the input is a layout of an underlying network, that is, a plane metro graph $(G, \mathcal{L})$ whose embedding is realized by a drawing Γ of G. The drawing Γ might be produced automatically by the methods presented in Chapter 3, manually by a graphic designer, or simply be given as the geographic layout. Each line ℓ in the line cover $\mathcal{L}$ is a simple path $\ell = (v_0, v_1, \ldots, v_k)$ in G. The vertices v_0 and v_k are called the *termini* of ℓ, the vertices $v_1, \ldots, v_{k-1}$ are called *intermediate stations*. Recall that $|\ell| = k$ is the length of ℓ. The *total edge size* of $\mathcal{L}$ is abbreviated as $N = \sum_{\ell \in \mathcal{L}} |\ell|$. Note that $N \in O(|\mathcal{L}| \cdot |V|)$ since each line is a simple path of at most $|V|$ vertices.

To simplify notation we assign arbitrary directions to the edges of G. We allow to access each directed edge uv both as uv, the forward edge from u to v, and as vu, the backward edge from v to u. Both notations refer to the same edge just from a different perspective. An edge uv is included in a line ℓ, in short $uv \in \ell$, if u and v are consecutive vertices in ℓ. The set of all lines that include an edge uv is denoted as $\mathcal{L}_{uv} = \mathcal{L}_{vu} = \{\ell \in \mathcal{L} \mid uv \in \ell\}$. Now for each edge $uv \in E$ we define two *line orders* $<^u_{uv}$ and $<^v_{uv}$ of $\mathcal{L}_{uv}$ in the endpoints of uv. For two lines ℓ_1 and ℓ_2 in $\mathcal{L}_{uv}$ we write $\ell_1 <^u_{uv} \ell_2$ (or $\ell_1 <^v_{uv} \ell_2$) if ℓ_1 is right of ℓ_2 at the endpoint u (or v) with respect to the direction of uv. Note that the orders are reversed if uv is accessed as the backward edge vu, that is, $\ell_1 <^u_{uv} \ell_2$ if and only if $\ell_2 <^u_{vu} \ell_1$. The sorted sequence of the lines in $\mathcal{L}_{uv}$ with respect to $<^u_{uv}$ is denoted as s^u_{uv}; analogously s^v_{uv} is the sorted sequence of lines with respect to $<^v_{uv}$. Again, the sequences s^u_{vu} and s^v_{vu} are the reversed sequences of s^u_{uv} and s^v_{uv}.

(a) With periphery condition. (b) Without periphery condition.

Figure 4.3: Layout of a terminus (middle vertex) with and without periphery condition. The layout without periphery condition introduces a gap between the ongoing lines (b).

A *line crossing* is a crossing between two lines ℓ_1 and ℓ_2 along a shared edge uv. The two lines cross on uv if $\ell_1 <_{uv}^{u} \ell_2$ and $\ell_2 <_{uv}^{v} \ell_1$ or vice versa. Abstracting from geometry, the number of line crossings along an edge uv is thus equal to the number of inversions between s_{uv}^{u} and s_{uv}^{v}.

One of our constraints is that line crossings shall not be hidden "below" a metro station. To that end we define a line order $<_{uv}^{v}$ to be *compatible* with the vertex v if the following holds. Apart from uv, let $vw_1, vw_2, \ldots, vw_k$ be the other edges incident to v in counterclockwise order starting from uv. We consider the sequence s_{uv}^{v} and the concatenated sequence $s' = \prod_{i=1}^{k} s_{vw_i}^{v}$. Then $<_{uv}^{v}$ is compatible with v if s_{uv}^{v} is a subsequence of s'. In other words, the lines that enter v through the edge uv and leave v through the edges $vw_1, vw_2, \ldots, vw_k$ do not change their relative order. We say that a vertex v is *admissible* if the line orders for all incident edges are compatible with v.

In the unconstrained *metro-line crossing minimization* problem (MLCM) the aim is to find for each edge $uv \in E$ two orders $<_{uv}^{u}$ and $<_{uv}^{v}$ of the lines in $\mathcal{L}_{uv}$ such that the number of line crossings is minimal and all vertices are admissible. A solution to MLCM is denoted as a *line layout*.

We also consider MLCM with *periphery condition* (MLCM-P), a variant that places a further constraint on the line orders in termini. So let v be a vertex and uv be one of its incident edges. The periphery condition requires that all lines in $\mathcal{L}_{uv}$ for which v is a terminus must be placed outermost in the order $<_{uv}^{v}$, that is, terminating lines are either placed first or last in the sequence s_{uv}^{v}. Figure 4.3 illustrates the effect of the periphery condition. This condition ensures that terminating lines do not introduce gaps in the course of the ongoing lines. Gaps between parallel lines disrupt the uniform appearance of the underlying edge and hence are to be avoided in order to improve readability. Moreover, line termini are important stations in the network. For example, trains or buses usually display the name of their final destination. Hence placing terminating lines outermost in their final destinations accentuates them for a simpler identification on the map. Many of the real-world maps in Ovenden's collection [Ove03] adhere to the periphery condition. It has recently been shown that choosing an optimal terminus assignment is NP-hard [BKPS08].

The MLCM-P problem gives rise to a closely related (but computationally feasible) variant that additionally specifies for each line on which side of its first and last edge the two termini must be placed. In that way we have a side assignment for each terminus of each line given as the input. We denote this variant as MLCM with periphery condition and terminus assignments (MLCM-PA). MLCM-PA occurs in situations, in which, for example, the physical location of the tracks or the bus stop of the terminating line in a terminus yields this information. Alternatively, the terminus assignment can be obtained from an ILP formulation that computes the optimal side assignment of terminating lines [AGM08].

Note that another interesting MLCM variant can be reduced to MLCM-PA: if all lines in $\mathcal{L}$ satisfy the property that they terminate at leaves of G, the unconstrained MLCM

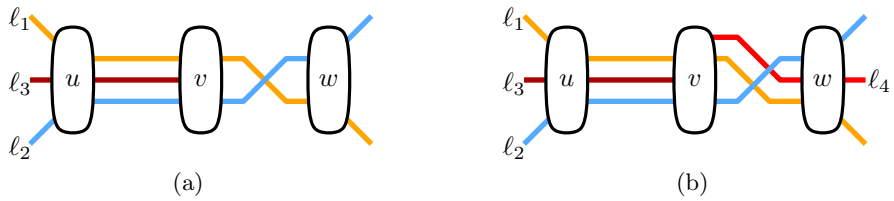


Figure 4.4: Lines ℓ_1 and ℓ_2 must cross along their common subpath (u, v, w). If line ℓ_3 is present, the ℓ_1-ℓ_2 crossing is optimally placed on edge vw (a). If ℓ_3 and ℓ_4 are present, a new crossing is added regardless of the position of the ℓ_1-ℓ_2 crossing (b).

problem for $(G, \mathcal{L})$ corresponds to MLCM-PA where all lines in a terminus are assigned to the same side. We call this version of the problem MLCM-T1. Since there are no ongoing lines in degree-1 vertices the periphery condition does not restrict the possible line orders if all lines are assigned to the same side. This variant has practical relevance, since in many real-world networks the lines lead from one peripheral degree-1 terminus through the city center, where interchanges are possible, to another peripheral degree-1 terminus in the suburbs.

To get a flavor of the basic MLCM problem without periphery condition, observe that the topology of G enforces some unavoidable line crossings. In Figure 4.4, lines ℓ_1 and ℓ_2 share the subpath (u, v, w), which ℓ_1 enters above ℓ_2 and leaves below ℓ_2. Hence the two lines must cross along (u, v, w) either on edge uv or on edge vw. If there is another line ℓ_3 that enters u between ℓ_1 and ℓ_2 and terminates at v, the crossing of ℓ_1 and ℓ_2 should be placed on edge vw to avoid additional line crossings (Figure 4.4a). But if another line ℓ_4 enters w between ℓ_1 and ℓ_2 and terminates at v, we can no longer avoid an additional line crossing by shifting the crossing of ℓ_1 and ℓ_2. Either ℓ_3 or ℓ_4 must cross one of ℓ_1 or ℓ_2 (Figure 4.4b). Note that in this situation the periphery condition would make each of ℓ_3 and ℓ_4 cross one of ℓ_1 and ℓ_2 in order to terminate outermost in v.

Contributions. We first consider the unconstrained MLCM problem. In Section 4.3 we give an $O(|I_{uv}|^2)$-time algorithm to solve the MLCM problem on a single edge uv of the metro graph, where $I_{uv} \subseteq \mathcal{L}_{uv}$ is the set of lines in $\mathcal{L}_{uv}$ that do not terminate in u or v. In this situation we assume that the orders $<^u_{uv}$ and $<^v_{uv}$ are already partially given by the compatibility constraint for vertices u and v. More precisely, all lines in $\mathcal{L}_{uv}$ are already ordered in their non-terminus stations. It only remains to determine the optimal positions of the terminating lines in their respective terminus. Clearly, our solution for the one-edge case is just a small step towards solving the general MLCM problem. Until today it is an open question whether the analogously defined two-edge MLCM problem is efficiently solvable or NP-hard. In Section 4.4 we sketch why an extension of the algorithm for one edge will not work correctly for two edges.

In the second part we study MLCM with periphery condition (Section 4.5). This problem is known to be NP-hard, even if G is a path [BKPS08]. For the MLCM-PA variant, however, in which all terminus assignments are specified in the input, we present an algorithm that computes in $O(|\mathcal{L}|^2 \cdot |V|)$ time an optimal line layout for general metro graphs $(G, \mathcal{L})$. This significantly improves a previous $O(|\mathcal{L}|^3 \cdot |E|^{2.5})$-time algorithm of Asquith et al. [AGM08]. Moreover, Argyriou et al. [ABKS09] recently gave an $O((|E| + |\mathcal{L}|^2) \cdot |E|)$-time algorithm for the MLCM-T1 problem. Since MLCM-T1 is a special case of MLCM-PA our algorithm improves the result of Argyriou et al. as well. Table 4.1 summarizes our results in the context of the results obtained in related work.

problem	graph class	restrictions	result	reference
MLCM	single edge uv	–	$O(\|I_{uv}\|^2)$	Theorem 4.2
	m-edge path	–	open	Section 4.4
MLCM-P	path	–	NP-hard	[BKPS08]
	plane graph	–	ILP + MLCM-PA	[AGM08]
MLCM-PA	path	2-side model	$O(\|\mathcal{L}\| \cdot \|V\|)$	[BKPS08]
	left-to-right tree	2-side model	$O(\|\mathcal{L}\| \cdot \|V\| \cdot \log d)$	[BKPS08]
	plane graph	–	$O(\|\mathcal{L}\|^3 \cdot \|E\|^{2.5})$	[AGM08]
	plane graph	2-side model	$O(\|V\| \cdot (\|E\| + \|\mathcal{L}\|))$	[ABKS09]
	plane graph	–	$O(\|\mathcal{L}\|^2 \cdot \|V\|)$	Theorem 4.3
MLCM-T1	left-to-right tree	2-side model	$O(\|\mathcal{L}\| \cdot \|V\| \cdot \log d)$	[BKPS08]
	plane graph	2-side model	$O(\|V\| \cdot (\|E\| + \|\mathcal{L}\|))$	[ABKS09]
	plane graph	–	$O((\|E\| + \|\mathcal{L}\|^2) \cdot \|E\|)$	[ABKS09]
	plane graph	–	$O(\|\mathcal{L}\|^2 \cdot \|V\|)$	Corollary 4.1

Table 4.1: Overview of results for the MLCM problem and its variants. Algorithmic results are given by their running time. Terms are used as introduced in Section 4.1, and d is the maximum indegree of the left-to-right tree.

4.2 Related Work

Our initial work, which presented the solution of MLCM for a single edge (see Section 4.3), introduced the MLCM problem in the literature and mentioned MLCM-P as an interesting open problem [BNUW07]. In the meantime, more results appeared in the literature as summarized in Table 4.1.

Bekos et al. [BKPS08] studied MLCM-P on path and tree networks. They proved that MLCM-P is NP-hard, even if the underlying graph G is a path. Motivated by the hardness of MLCM-P they introduced the MLCM-PA variant, for which they gave efficient algorithms for certain restricted classes of underlying graphs in the so-called *2-side* model. In the 2-side model all vertices are drawn as rectangles and the edges can be connected only to the left and right sides of the vertex rectangles. Their first algorithm computes in $O(|\mathcal{L}| \cdot |V|)$ time an optimal line layout for the case that G is a path. In the second part of their paper, Bekos et al. considered MLCM-T1 for a graph class called *left-to-right tree structured networks*. In a left-to-right tree structured network the underlying graph G is drawn in the 2-side model, edges are directed from left to right, and all lines are x-monotone paths in G. For this class of metro graphs they gave an $O(|\mathcal{L}| \cdot |V| \cdot \log d)$-time algorithm, where d is the maximum indegree in G. The algorithm performs a tree traversal and computes the line orders based on an appropriate order of their termini. Using a simple reduction rule their algorithm can also be applied to MLCM-PA instances on left-to-right tree structured networks. The reduction adds dummy degree-1 vertices that extend all terminating lines beyond their proper termini and then applies the MLCM-T1 algorithm.

Asquith et al. [AGM08] took a different approach to the MLCM-P problem. They formulated an ILP that determines an optimal side assignment for the line termini. For all pairs of lines they determined the (possibly many) common subpaths and formulated a set of crossing rules that determine whether a pair of lines crosses along a common subpath. These rules are created in $O(|\mathcal{L}|^2 \cdot |E|)$ time and converted into the constraints

of an ILP. Note that in the worst case it takes exponential time to solve this ILP. Once the side assignment is fixed by the ILP, the problem turns into an MLCM-PA instance and it remains to determine the line orders along each edge. Reusing a circuit-board layout algorithm by Marek-Sadowska and Sarrafzadeh [MSS95], Asquith et al. solved the final MLCM-PA step in $O(|\mathcal{L}|^3 \cdot |E|^{2.5})$ time. As an alternative to the ILP, the authors also suggested a heuristic that takes a local view on the line orders. They reported that in initial experiments the heuristic solutions could differ significantly from the optimum depending on how nested the metro lines were.

Recently, Argyriou et al. [ABKS09] continued the earlier work of Bekos et al. [BKPS08]. They extended the 2-side model to the more general k-side model and also considered general planar underlying graphs instead of only paths and trees. For MLCM-T1 in the general k-side model they presented an $O((|E| + |\mathcal{L}|^2) \cdot |E|)$-time algorithm. Additionally, for $k = 2$, they improved the above running time to $O(|V| \cdot (|E| + |\mathcal{L}|))$. The latter algorithm is also able to solve MLCM-PA and thus it improves the running time of the algorithm of Asquith et al. [AGM08] for graphs in the 2-side model.

4.3 Line Layout for a Single Edge

In this section we make a first step towards the general MLCM problem by restricting our attention to a single edge of the underlying network. We first introduce some notation.

Let $e = uv$ be an edge of the underlying graph G. We split the set $\mathcal{L}_{uv}$ of lines that include uv into three subsets:

- T_u is the set of lines that terminate in u along uv,
- T_v is the set of lines that terminate in v along uv,
- I_{uv} is the set of lines for which both u and v are intermediate stations.

We may assume that $T_u \cap T_v = \emptyset$ since lines that consist of the single edge uv can always be placed topmost without causing any line crossings. Hence T_u, T_v, and I_{uv} is a partition of $\mathcal{L}_{uv}$.

We further assume that the suborder of $I_{uv} \cup T_v$ in $<^u_{uv}$ is given, that is, the lines that pass through u are already ordered. Similarly, the suborder of $I_{uv} \cup T_u$ in $<^v_{uv}$ is given. These suborders are induced by the line orders of the other edges incident to u and v (excluding uv) since the line orders $<^u_{uv}$ and $<^v_{uv}$ must be compatible with u and v. For ease of notation we will abbreviate the orders $<^u_{uv}$ and $<^v_{uv}$ by $<^u$ and $<^v$ in this section if the context of the edge uv is clear. The remaining layout problem is as follows.

Problem 4.1 (One-Edge Layout) *Given a metro graph $(G, \mathcal{L})$ and a distinguished edge uv, complete the partially specified line orders $<^u$ and $<^v$ by inserting the lines $\ell \in T_u$ into $<^u$ and the lines $\ell \in T_v$ into $<^v$ such that the number of line crossings along uv is minimized.*

In contrast to the well-known NP-hard problem of minimizing crossings in a two-layer bipartite graph [GJ83], in which the vertices of the bipartition are to be ordered on two parallel lines such that the edges drawn as straight-line segments have a minimum number of crossings, the one-edge layout problem is polynomially solvable. The main reason is that there is an optimal layout of $\mathcal{L}_{uv}$ such that no two lines in T_u intersect and no two lines in T_v intersect. This observation allows us to split the problem and to apply dynamic

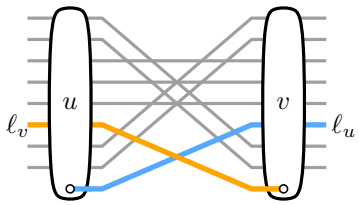


Figure 4.5: The lines in I_{uv} are drawn in gray, the lines in T_v in yellow, and the lines in T_u in blue. In an optimal solution the terminating lines $\ell_v \in T_v$ and $\ell_u \in T_u$ intersect.

programming. It is then rather easy to come up with an $O(n^5)$-time algorithm and with some effort we reduce the running time to $O(n^2)$, where $n = |I_{uv}|$.

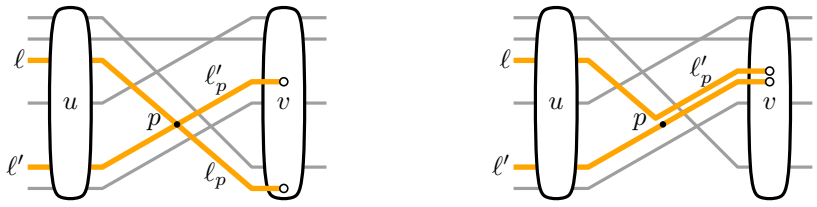
Observe that the orders $<^u$ and $<^v$ already determine the number cr_{uv} of crossings between pairs of lines in I_{uv} and that the lines in T_v inserted into $<^v$ and the lines in T_u inserted into $<^u$ do not change cr_{uv}. Thus, there is no need to take crossings between lines in I_{uv} into account any more. On the other hand, inserting the remaining lines into the line orders affects the number of crossings between lines in $T_v \cup T_u$ and lines in I_{uv} and the number of crossings between lines in T_v and lines in T_u in a non-trivial way. Figure 4.5 shows that a line $\ell_v \in T_v$ can indeed cross a line $\ell_u \in T_u$ in the unique optimal solution. Throughout this section lines in I_{uv} are drawn in gray, lines in T_v in yellow and lines in T_u in blue. We will now show that no two lines in T_v (and analogously in T_u) cross in an optimal solution. This nice property is the key for solving the one-edge layout problem in polynomial time.

Lemma 4.1 *In any optimal solution for the one-edge layout problem no pair of lines in T_v and no pair of lines in T_u intersects.*

Proof. Assume to the contrary that there is an optimal solution σ with a pair of lines in T_v that intersects. Among all pairs of lines that intersect in σ let $\{\ell, \ell'\}$ be the one whose intersection point p is rightmost. Without loss of generality, $\ell' <^u \ell$. Let ℓ_p and ℓ'_p be the parts of ℓ and ℓ' to the right of p, see Figure 4.6a. Since σ is crossing minimal, ℓ_p and ℓ'_p intersect the minimum number of lines in $I_{uv} \cup T_u$ in order to get from p to v. In particular, the number of crossings between ℓ_p and lines of $I_{uv} \cup T_u$ and between ℓ'_p and lines of $I_{uv} \cup T_u$ must be the equal; otherwise we could place ℓ_p parallel to ℓ'_p (or vice versa) and thus reduce the number of crossings. Since, however, the number of crossings to the right is the same we can easily get rid of the crossing between ℓ and ℓ' by replacing ℓ_p by a copy of ℓ'_p infinitesimally close above ℓ'_p, see Figure 4.6b. This contradicts the optimality of σ. The proof for a pair of lines in T_u is analogous. □

We now assume that no two lines of T_v are consecutive in $<^u$ and analogously no two lines of T_u are consecutive in $<^v$. This does not restrict the general problem since bundles of consecutive lines can always be drawn parallelly in an optimal layout. Thus a single representative line suffices to determine the optimal layout for the whole bundle. Technically, for a bundle of k consecutive lines we assign the weight k to the representative line. Our algorithm will then run in a weighted fashion that counts $k \cdot k'$ crossings for a crossing of two lines with weights k and k'. For ease of presentation we explain only the unweighted problem here.

Let n, n_v, and n_u be the number of lines in I_{uv}, T_v, and T_u, respectively. Note that by the above assumption we obtain $n_u \leq n + 1$ and $n_v \leq n + 1$.



(a) Lines ℓ and ℓ' intersect in p.

(b) The crossing of ℓ and ℓ' is removed.

Figure 4.6: Two lines of T_v do not intersect in an optimal solution.

Recall that by assumption the lines in $I_{uv} \cup T_v$ are already ordered by $<^u$ and the lines in $I_{uv} \cup T_u$ are already ordered by $<^v$. We denote the ordered sequence of the $n + n_v$ lines in $I_{uv} \cup T_v$ by $S_u = (s_1^u, s_2^u, \ldots, s_{n+n_v}^u)$, where $s_i^u <^u s_{i+1}^u$ for $i = 1, \ldots, n + n_v - 1$. In our figures we stick to the convention that the edge uv is directed from left to right and thus S_u is the bottom-to-top sequence of $I_{uv} \cup T_v$. Analogously, we obtain the sequence $S_v = (s_1^v, s_2^v, \ldots, s_{n+n_u}^v)$ of the $n + n_u$ lines in $I_{uv} \cup T_u$ ordered from bottom to top by $<^v$. A line ℓ in T_u can terminate below s_1^u, between two neighboring lines s_i^u and s_{i+1}^u, or above $s_{n+n_v}^u$. We define the *position index* of ℓ as the index of the lower of the two lines and as 0 if ℓ is placed below s_1^u. The position index of a line $\ell \in T_v$ is defined analogously. Let $S_u|T_v = (s_{\mu(1)}^u, s_{\mu(2)}^u, \ldots, s_{\mu(n_v)}^u)$ denote the ordered subsequence of S_u of the lines in T_v, and let $S_v|T_u = (s_{\pi(1)}^v, s_{\pi(2)}^v, \ldots, s_{\pi(n_u)}^v)$ denote the ordered subsequence of S_v of the lines in T_u. Here, μ and π are injective functions that filter the lines T_v out of all ordered lines $I_{uv} \cup T_v$ in S_u and the lines T_u out of all ordered lines $I_{uv} \cup T_u$ in S_v, see Figure 4.7.

4.3.1 Preprocessing

When fixing the course of a line in T_u (or T_v), that is, inserting it at some position i into the order $<^u$ (or $<^v$), we need to compute the number of crossings induced by this position. So let $\ell = s_{\pi(j)}^v \in T_u$ be a line that has position index $\pi(j)$ in S_v. We assign the position index i in S_u to ℓ. Then the number of crossings between ℓ and all lines in I_{uv} is denoted by $\mathrm{cr}_u(i, j)$. This number is determined as follows. The line ℓ crosses a line $\ell' \in I_{uv}$ with left index i' and right index j' if and only if either it holds that $i' \leq i$ and $j' > \pi(j)$ or it holds that $i' > i$ and $j' < \pi(j)$. The table cr_u for all lines in T_u has $(n + n_v + 1) \times n_u = O(n^2)$ entries. For a fixed position i in S_u we can compute the table row $\mathrm{cr}_u(i, \cdot)$ as follows. We start with $j = 1$ and compute the number of lines in I_{uv} that intersect the line ℓ with indices i and $\pi(j)$. Then, we increment j and obtain $\mathrm{cr}_u(i, j + 1)$ as $\mathrm{cr}_u(i, j)$ minus the number of lines in I_{uv} that are no longer intersected plus the lines that become newly intersected. As any of the n lines in I_{uv} receives the status "no longer intersected" or "newly intersected" at most once and this status can easily be checked by scanning S_v, this takes linear time per row. Thus, in total we can compute the table cr_u in $O(n^2)$ time.

We define $\mathrm{cr}_v(i, j)$ analogously to be the number of crossings of the lines in I_{uv} with the line in T_v that has index $\mu(i)$ in S_u and position j in S_v. Computing cr_v is analogous to to computing cr_u and also take $O(n^2)$ time.

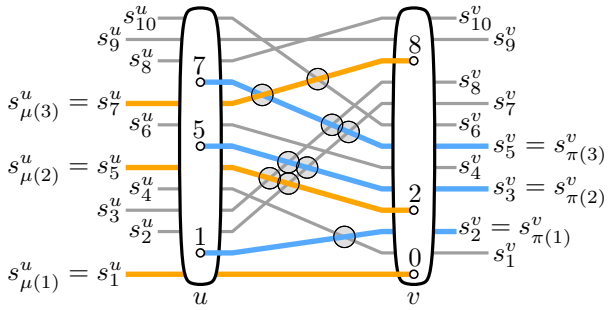


Figure 4.7: The orders S_u and S_v and the induced suborder $S_u|T_v$ and $S_v|T_u$. This configuration corresponds to $F(7,3)$: the topmost blue line $s^v_{\pi(3)}$ terminates at position 7 in vertex u.

4.3.2 Dynamic Programming Algorithm

Assume that we fix the destination of $s^v_{\pi(j)}$ to some position $i \in \{0,\dots,n+n_u\}$. Then we define $F(i,j)$ as the minimum number of crossings of

(a) the lines in $T_v \cap \{s^u_1,\dots,s^u_i\}$ with the lines in $I_{uv} \cup T_u$ and

(b) the lines in $T_u \cap \{s^v_1,\dots,s^v_{\pi(j)}\}$ with the lines in $I_{uv} \cup T_v$.

An example is depicted in Figure 4.7, where the nine crossings indicated by gray disks are counted in $F(i,j)$ for $i=7$ and $j=3$. The values $F(i,j)$ define an $(n+n_v+1)\times n_u$-table F.

Once the last column $F(\cdot,n_u)$ of the table F is computed, that is, the positions for all lines in T_u are fixed, we can determine the optimal solution for $\mathcal{L}_{uv}$ as

$$F^* := \min\{F(i,n_u) + C(i,n+n_v,n_u+1) \mid i = 0,\dots,n+n_v\},$$

where $C(i,n+n_v,n_u+1)$ is the remaining number of crossings of the lines in $T_v \cap \{s^u_{i+1},\dots,s^u_{n+n_v}\}$, that is, the lines in T_v with a position index larger than i in S_u, with lines in $I_{uv} \cup T_u$; these crossings are not yet counted in $F(i,n_u)$.

Before describing the recursive computation of $F(i,j)$ we introduce another notation. Let's assume that $s^v_{\pi(j-1)}$ terminates at position k and $s^v_{\pi(j)}$ terminates at position i, where $0 \le k \le i \le n+n_v$ and $j \in \{1,\dots,n_u\}$. Then let $C(k,i,j)$ denote the minimum number of crossings that the lines $T^{k,i}_v := T_v \cap \{s^u_{k+1},\dots,s^u_i\}$ cause with the lines in $I_{uv} \cup T_u$. In other words, $C(k,i,j)$ counts the minimal number of crossings of all lines of T_v in the interval between the positions k and i, which are the positions of the two lines $s^v_{\pi(j-1)}$ and $s^v_{\pi(j)}$ in S_u. This situation is illustrated in Figure 4.8, where the four crossings of the yellow lines between positions k and i that are marked with gray disks are counted in the term $C(k,i,j)$. We store all values $C(k,i,j)$ in a three dimensional table C. Analogously to the definition of $T^{k,i}_v$, we define $T^{k,i}_u := T_u \cap \{s^v_{k+1},\dots,s^v_i\}$. The following theorem gives the recursion for F and shows its correctness.

Theorem 4.1 *The values $F(i,j)$, $i = 0,\dots,n+n_v$, $j = 1,\dots,n_u$, can be computed recursively by*

$$F(i,j) = \begin{cases} \min_{k\le i}\{F(k,j-1) + C(k,i,j) + cr_u(i,j)\} & \text{if } i \ge 1, j \ge 2 \\ \sum_{l=1}^{j} cr_u(0,l) & \text{if } i = 0, j \ge 1 \\ C(0,i,1) + cr_u(i,1) & \text{if } i \ge 1, j = 1. \end{cases} \tag{4.1}$$

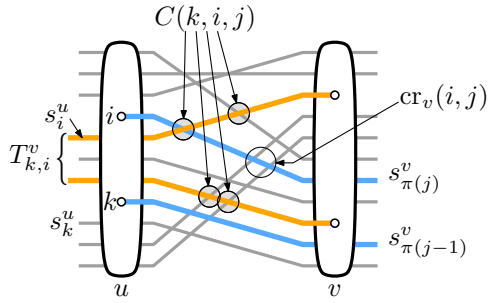


Figure 4.8: Situation for computing $F(i,j)$: The blue lines $s_{\pi(j-1)}^v$ and $s_{\pi(j)}^v$ terminate at positions k and i, respectively. The term $C(k,i,j)$ denotes the minimum number of crossings of the yellow lines $T_v^{k,i}$ with the lines in $I_{uv} \cup T_u$, here $C(k,i,j) = 4$. The term $\mathrm{cr}_u(i,j)$ counts the number of crossings of $s_{\pi(j)}^v$, terminating at position i, with the lines in I_{uv}, here $\mathrm{cr}_u(i,j) = 2$.

Proof. The base cases of (4.1) consist of two parts. In the first row $F(0,\cdot)$, an entry $F(0,j)$ means that all lines $s_{\pi(1)}^v, \ldots, s_{\pi(j)}^v$ terminate at position 0 in u and hence the required number of crossings is just the number of crossings of these lines with I_{uv}, which equals the sum $\sum_{l=1}^{j} \mathrm{cr}_u(0,l)$ as given in (4.1). In the first column $F(\cdot,1)$, an entry $F(i,1)$ reflects the situation that line $s_{\pi(1)}^v$ terminates at position i. The required number of crossings in this case is simply $\mathrm{cr}_u(i,1)$, the number of crossings of $s_{\pi(1)}^v$ with I_{uv}, plus $C(0,i,1)$, the number of crossings of $T_v^{0,i}$ with $I_{uv} \cup T_u$ as given in (4.1).

The general case of the recursion in (4.1) means that the value $F(i,j)$ can be composed of the optimal placement $F(k,j-1)$ of the lines in T_u below and including $s_{\pi(j-1)}^v$ (which itself terminates at some position $k \leq i$), the number $C(k,i,j)$ of crossings of lines in $T_v^{k,i}$ with $I_{uv} \cup T_u$, and the number of crossings $\mathrm{cr}_u(i,j)$ of $s_{\pi(j)}^v$ placed at position i.

We prove the correctness of the general case by induction. Due to Lemma 4.1 we know that $s_{\pi(j)}^v$ cannot terminate below $s_{\pi(j-1)}^v$ in an optimal solution. Hence, for $s_{\pi(j)}^v$ terminating at position i, we know that $s_{\pi(j-1)}^v$ terminates at some position $k \leq i$. For each k we know by induction hypothesis that $F(k,j-1)$ is the correct minimum number of crossings as defined above. In order to extend the configuration corresponding to $F(k,j-1)$ with the next line $s_{\pi(j)}^v$ in T_u we need to add two terms: (a) the number of crossings of $T_v^{k,i}$ with $I_{uv} \cup T_u$, which is exactly $C(k,i,j)$, and (b) the number $\mathrm{cr}_u(i,j)$ of crossings that the line $s_{\pi(j)}^v$ (terminating at position i) has with I_{uv}. Note that potential crossings of $s_{\pi(j)}^v$ with lines in $T_v^{k,i}$ are already considered in the term $C(k,i,j)$. Figure 4.8 illustrates this situation: $s_{\pi(j)}^v$ is placed at position i in S_u, and $s_{\pi(j-1)}^v$ terminates at position k. The crossings of the configuration corresponding to $F(i,j)$ that are not yet counted in $F(k,j-1)$, are the $C(k,i,j)$ crossings of the yellow lines in $T_v^{k,i}$ (highlighted by gray disks) and the $\mathrm{cr}_u(i,j)$ encircled crossings of $s_{\pi(j)}^v$ with I_{uv}.

Finally, we have to show that taking the minimum value of the sum in the first case of (4.1) for all possible terminus positions k of line $s_{\pi(j-1)}^v$ yields an optimal solution for $F(i,j)$. Assume to the contrary that there is a better solution that yields $F'(i,j) < F(i,j)$ crossings. This solution induces a solution $F'(k',j-1)$, in which k' is the position of $s_{\pi(j-1)}^v$ in S_u. From Lemma 4.1 it follows that $k' \leq i$ and hence $s_{\pi(j-1)}^v$ runs completely below $s_{\pi(j)}^v$. Therefore we have $F'(k',j-1) \leq F'(i,j) - C(k',i,j) - \mathrm{cr}_u(i,j) < F(i,j) - C(k',i,j) - \mathrm{cr}_u(i,j) \leq F(k',j-1)$. But $F'(k',j-1) < F(k',j-1)$ is a contradiction to the optimality of $F(k',j-1)$ that we get from the induction hypothesis. □

If we store in each cell $F(i,j)$ a pointer to the respective predecessor cell $F(k,j-1)$ that minimizes (4.1) we can reconstruct the optimal line layout: starting at the cell $F(i,n_u)$ that minimizes F^*, we can reconstruct the genesis of the optimal solution using backtracking. Obviously, using the combinatorial solution to place all endpoints of $\mathcal{L}_{uv}$ in the correct order and then connecting them with straight-line segments results in a layout that has exactly F^* crossings in addition to cr_{uv}, the fixed number of crossings of I_{uv}.

Now, we can give a first, naive implementation: as mentioned earlier the tables cr_v and cr_u can be computed in $O(n^2)$ time. For the computation of one cell entry $C(k,i,j)$ we have to consider the at most n lines $T_v^{k,i}$ and their possible $n+1$ terminus positions in S_v. Once we have fixed the terminus position l of a line $\ell \in T_v^{k,i}$, we have to compute the number of crossings that ℓ has with $I_{uv} \cup T_u$. The crossings with I_{uv} are available from the table cr_v. For the crossings with T_u it is sufficient to distinguish the following three cases depending on the position index l of ℓ in S_v because we know that k is the terminus position of $s^v_{\pi(j-1)}$ and i is the terminus position of $s^v_{\pi(j)}$. If $l \geq \pi(j)$ then ℓ intersects all lines in $T_u^{\pi(j)-1,l}$, if $\pi(j-1) \leq l < \pi(j)$ then ℓ does not intersect any line in T_u, and if $l < \pi(j-1)$ then ℓ intersects all lines in $T_u^{l,\pi(j-1)}$. Thus, computing one of the $O(n^3)$ cells of the table C requires $O(n^2)$ time, so in total we need $O(n^5)$ time for filling C. This dominates the naive $O(n^3)$-time computation of the table F. Next we show how to speed up the computation of F and C.

4.3.3 Improving the Running Time

For the moment let's assume that the values $C(k,i,j)$ and $\mathrm{cr}_u(i,j)$ are available in constant time. Then the computation of the $(n+n_v+1)\times n_u$-table F still needs $O(n^3)$ time because the minimum in (4.1) is over a set of $O(n)$ elements. The following series of lemmas shows how to reduce the running time for computing F to $O(n^2)$. First we show a property of the entries of the table C.

Lemma 4.2 *The table C is additive in the sense that $C(k,i,j) = C(k,l,j) + C(l,i,j)$ for $k \leq l \leq i$.*

Proof. Since $C(k,i,j)$ denotes the number of crossings of the lines in $T_v^{k,i}$ and no two of these lines intersect each other in an optimal layout by Lemma 4.1, we can split the layout corresponding to $C(k,i,j)$ at any position l, $k \leq l \leq i$ and get two (possibly non-optimal) configurations for the induced subproblems. This implies $C(k,i,j) \geq C(k,l,j) + C(l,i,j)$.

Conversely, we can get a configuration for $C(k,i,j)$ by putting together the optimal solutions of the subproblems. Without loss of generality, this introduces no additional crossings; otherwise they could be removed as in the proof of Lemma 4.1. Hence we have $C(k,i,j) \leq C(k,l,j) + C(l,i,j)$. □

The next lemma shows that it is not necessary to compute all entries of C in order to compute the table F.

Lemma 4.3 *Given the table C, the table F can be computed in $O(n^2)$ time.*

Proof. Having computed entry $F(i-1,j)$ we can compute $F(i,j)$ in constant time as follows:

$$F(i,j) = \min \begin{cases} F(i-1,j) + C(i-1,i,j) - \mathrm{cr}_u(i-1,j) + \mathrm{cr}_u(i,j), \\ F(i,j-1) + \mathrm{cr}_u(i,j). \end{cases} \tag{4.2}$$

The correctness follows from (4.1), Lemma 4.2, and the fact that $C(i,i,j)$ vanishes:

$$\begin{aligned}
F(i,j) &\overset{(4.1)}{=} \min \begin{cases} \min\limits_{k<i}\{F(k,j-1)+C(k,i,j)+\mathrm{cr}_u(i,j)\}, \\ F(i,j-1)+C(i,i,j)+\mathrm{cr}_u(i,j) \end{cases} \\
&\overset{\text{Lemma 4.2}}{=} \min \begin{cases} \min\limits_{k\le i-1}\{F(k,j-1)+C(k,i-1,j)+C(i-1,i,j)+\mathrm{cr}_u(i,j)\}, \\ F(i,j-1)+\mathrm{cr}_u(i,j) \end{cases} \\
&\overset{(4.1)}{=} \min \begin{cases} F(i-1,j)-\mathrm{cr}_u(i-1,j)+C(i-1,i,j)+\mathrm{cr}_u(i,j), \\ F(i,j-1)+\mathrm{cr}_u(i,j). \end{cases}
\end{aligned}$$

In the first column $F(\cdot,1)$, we can reformulate the recursion for $i \geq 1$ as follows:

$$\begin{aligned}
F(i,1) &\overset{(4.1)}{=} C(0,i,1)+\mathrm{cr}_u(i,1) \\
&\overset{\text{Lemma 4.2}}{=} C(0,i-1,1)+C(i-1,i,1)+\mathrm{cr}_u(i,1) \\
&\overset{(4.1)}{=} F(i-1,1)-\mathrm{cr}_u(i-1,1)+C(i-1,i,1)+\mathrm{cr}_u(i,1)
\end{aligned}$$

Hence the whole table F can be computed in $O(n^2)$ time. □

Observe that due to the reformulation in Lemma 4.3 we need only the values $C(i-1,i,j)$ explicitly in order to compute F. Next, we show that we can compute these relevant values in $O(n^2)$ time. To simplify notation we abbreviate $C(i-1,i,j)$ as $C'(i,j)$.

Lemma 4.4 *The values $C'(i,j)$, $i=1,\ldots,n+n_v$, $j=1,\ldots,n_u$ can be computed in $O(n^2)$ time.*

Proof. We compute $C'(i,j)$ row-wise, that is, we fix a row index i and increase the column index j. Recall that $C(k,i,j)$ was defined as the minimum number of crossings of the lines in $T_v^{k,i}$ with the lines in $I_{uv} \cup T_u$ under the condition that $s^v_{\pi(j-1)}$ terminates at position k and $s^v_{\pi(j)}$ terminates at position i. For $C'(i,j) = C(i-1,i,j)$ this means we have to consider crossings of the set $T_v^{i-1,i} = \{s_i^u\} \cap T_v$. Hence, we distinguish two cases: either $s_i^u \in I_{uv}$ and then $T_v^{i-1,i}$ is empty or $s_i^u \in T_v$ and we have to place the line s_i^u optimally. Obviously, in the first case we have $C'(i,j)=0$ for all j as there is no line to place in $T_v^{i-1,i}$.

So consider the case that $s_i^u \in T_v$. For each j we split the set of candidate terminus positions for s_i^u into the three intervals $[0,\pi(j-1))$, $[\pi(j-1),\pi(j))$, and $[\pi(j),n+n_u]$. Let $\mathrm{LM}(i,j)$, $\mathrm{MM}(i,j)$, and $\mathrm{UM}(i,j)$ denote the minimum number of crossings of s_i^u with $I_{uv} \cup T_u$ for terminus positions in $[0,\pi(j-1))$, $[\pi(j-1),\pi(j))$, and $[\pi(j),n+n_u]$, respectively. Then we have

$$C'(i,j) = \min\{\mathrm{LM}(i,j), \mathrm{MM}(i,j), \mathrm{UM}(i,j)\}, \tag{4.3}$$

the minimum of the three disjoint position intervals. The situation is illustrated in Figure 4.9.

Next, we show how to compute MM, LM, and UM. First, we consider MM. Recall that $\mathrm{cr}_v(i,j)$ was defined as the number of crossings of the lines in I_{uv} with the line $s^u_{\mu(i)}$ that terminates at position j in v. It follows that

$$\mathrm{MM}(i,j) = \min\{\mathrm{cr}_v(\mu^{-1}(i),k) \mid \pi(j-1) \le k < \pi(j)\}, \tag{4.4}$$

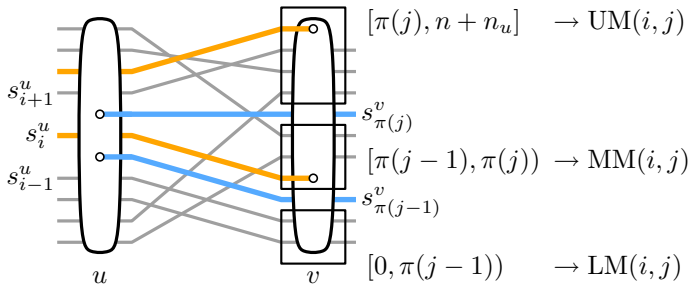


Figure 4.9: Splitting the candidate terminus positions for $s^u_i \in T_v$ into three intervals with respect to $\pi(j-1)$ and $\pi(j)$.

where $\pi(0)$ is defined as 0. Because of Lemma 4.1 there are no lines of T_u intersecting the funnel-shaped region between $s^v_{\pi(j-1)}$ and $s^v_{\pi(j)}$. Hence, if the line s^u_i terminates at position $k \in [\pi(j-1), \pi(j))$ it does not cross any line of T_u and the definition of MM in (4.4) is correct. We can calculate $\mathrm{MM}(i,j)$ by a straight-forward minimum computation through $j = 1, \ldots, n_u$ which takes $O(n)$ time for each value of i and $O(n^2)$ time in total.

Secondly, we consider LM. Initially, in the case that $j = 1$ there is no line $s^v_{\pi(j-1)}$ and the corresponding interval is empty. Hence we set $\mathrm{LM}(i,1) = \infty$. Then we recursively compute

$$\mathrm{LM}(i, j+1) = \min\{\mathrm{LM}(i,j) + 1, \mathrm{MM}(i,j) + 1\}. \tag{4.5}$$

Observe that for $\mathrm{LM}(i, j+1)$ we merge the previous intervals corresponding to $\mathrm{LM}(i,j)$ and $\mathrm{MM}(i,j)$. Moreover the line $s^v_{\pi(j)}$, which previously ended at position i, now terminates at position $i-1$. Hence, in order to reach its terminus position in the interval $[0, \pi(j))$, the line s^u_i has to cross $s^v_{\pi(j)}$ in addition to the crossings counted before by $\mathrm{MM}(i,j)$ and $\mathrm{LM}(i,j)$. This explains the recursion in (4.5). The computation again requires $O(n)$ time for each value of i and $O(n^2)$ time in total.

Finally, we initialize $\mathrm{UM}(i, n_u) = 1 + \min\{\mathrm{cr}_v(\mu^{-1}(i), k) \mid \pi(n_u) \leq k \leq n + n_u\}$ as for $j = n_u$ the line s^u_i crosses the line $s^v_{\pi(n_u)}$ but no other line of T_u. In decreasing order we compute

$$\mathrm{UM}(i, j-1) = \min\{\mathrm{UM}(i,j) + 1, \mathrm{MM}(i,j) + 1\} \tag{4.6}$$

analogously to LM, which again requires $O(n)$ time for each i and $O(n^2)$ time in total.

Since the values $\mathrm{LM}(i,j)$, $\mathrm{MM}(i,j)$, and $\mathrm{UM}(i,j)$ are computed in $O(n^2)$ time the table C' is also computed in $O(n^2)$ time according to (4.3). □

Putting the intermediate results of Lemmas 4.3 and 4.4 together, we conclude:

Theorem 4.2 *The one-edge layout problem (Problem 4.1) can be solved in $O(n^2)$ time.*

In the implementation described so far, the algorithm requires $O(n^2)$ space to store the tables F, C', cr_u, and cr_v. Since n, the number of lines using an edge, is typically small this is well within the capacity of current machines. Nonetheless, the space requirements can be improved. If we are interested only in the minimum *number* of crossings (and not the actual layout) the required space can easily be reduced to $O(n)$ space as all tables can be computed row-wise: in F we need only two consecutive rows at a time and we can discard previous rows; in the other tables the rows are independent and can be computed on demand. This does not affect the time complexity. To restore the optimal layout, however, we need the pointers in F to do backtracking and hence we cannot easily discard rows of the table. Still,

we can reduce the required space to $O(n)$ with a method similar to a divide-and-conquer version of the Needleman-Wunsch algorithm for sequence alignment [DEKM98, Chapter 2] at the cost of a factor of 2 in the time complexity. Basically, the idea is to compute the pointerless table F in linear space as before. For $j > \lfloor n_u/2 \rfloor$, however, that is, for the columns in the right half of F, we maintain pointers from the entries in the current column $F(\cdot, j)$ to the cell in column $F(\cdot, \lfloor n_u/2 \rfloor)$ that is on the optimal backtracking path. Clearly, the number of pointers is only linear. Once we have found the minimal number of crossings $F^\star = F(i, n_u)$ for some i, we follow the pointer to find the entry $F(j, \lfloor n_u/2 \rfloor)$ which lies on the backtracking path. Now we can recursively solve the two subproblems in the upper left part of the table F from $F(0,0)$ to $F(j, \lfloor n_u/2 \rfloor)$ and in the lower right part from $F(j, \lfloor n_u/2 \rfloor)$ to $F(i, n_u)$, which together have half the size of F. Hence the optimal layout can be reconstructed in twice the time of the original algorithm.

4.4 Line Layout for a Path

Metro-line crossing minimization for a single edge of the underlying network G, as studied in the previous section, is clearly just a first step towards solving the MLCM problem in more general graphs. The next step is to consider MLCM on a path of length $m \geq 2$. Recall that MLCM-P is NP-hard for a path [BKPS08]. It would come as no surprise if the same was true for MLCM; as of today this question is still open, even for $m = 2$. In this section we give a hint as to why a dynamic programming based approach as for the one-edge layout will not work.

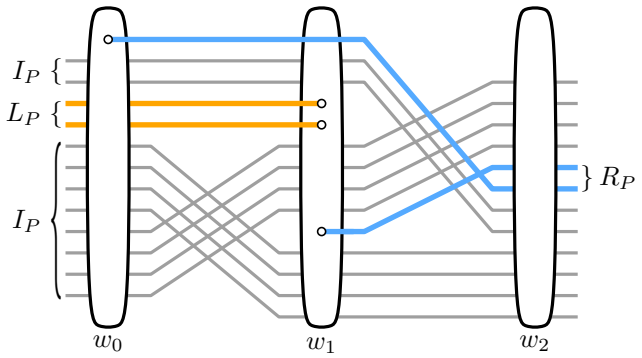
We first define the path-layout problem. Let $P = (u = w_0, w_1, \ldots, w_{m-1}, w_m = v)$ be a simple (left-to-right) m-edge path in G. We define $\mathcal{L}_P = \bigcup_{i=0}^{m-1} \mathcal{L}_{w_i w_{i+1}}$ as the set of all lines in $\mathcal{L}$ that use an edge of P. The set $\mathcal{L}_P$ is split into three subsets analogously to the one-edge case:

- $L_P = \bigcup_{i=0}^{m-1} \{\ell \in \mathcal{L}_{w_i w_{i+1}} \mid \ell \text{ terminates in } w_{i+1}\}$ is the set of lines that terminate along P coming from the left,
- $R_P = \bigcup_{i=0}^{m-1} \{\ell \in \mathcal{L}_{w_i w_{i+1}} \mid \ell \text{ terminates in } w_i\}$ is the set of lines that terminate along P coming from the right,
- $I_P = \mathcal{L}_P \setminus (L_P \cup R_P)$ is the set of lines that do not terminate along P.

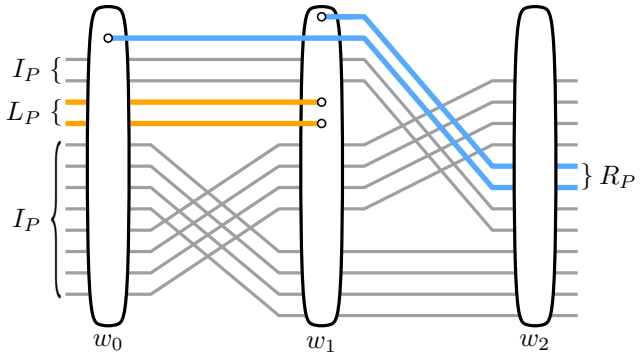
Finally, we assume that the position of all lines that enter P through an edge $v'w_i$, where $v' \notin P$ and $w_i \in P$, is known in the incoming line order $<_{v'w_i}^{w_i}$ of w_i—and thus by the compatibility requirement also in the other line orders of w_i.

Problem 4.2 (Path Layout) *Given a metro graph $(G, \mathcal{L})$ and a simple path $P = (w_0, \ldots, w_m)$ in G, determine all line orders $<_{w_i w_{i+1}}^{w_i}$ and $<_{w_i w_{i+1}}^{w_{i+1}}$ for $i = 0, \ldots, m-1$ such that the number of line crossings along P is minimized.*

We tried to apply the same dynamic-programming approach as for the one-edge case. The dilemma is, however, that the generalized version of Lemma 4.1 does not hold, namely that no two lines in L_P and no two lines in R_P intersect. Thus, a problem instance can no longer be separated into two independent subproblems along the lines in L_P and R_P, which seems to forbid dynamic programming. Figure 4.10 shows a 2-edge path for which the two lines in R_P (drawn in blue) must cross each other in the optimal solution, which



(a) Optimal solution (7 crossings).



(b) Non-optimal solution (8 crossings).

Figure 4.10: Two-edge path P in which two lines in R_P (blue) cross in the optimal solution (a). Any line layout without the blue-blue crossing has at least 8 crossings (b).

has seven relevant crossings (ignoring unavoidable crossings). Note that in any optimal solution the two yellow lines in L_P push the eight unavoidable crossings of the two topmost and the four bottommost lines in I_P (drawn in gray) to the edge w_1v. It is easy to verify that any line layout in which the two lines in R_P do not cross has at least eight crossings; one such example is shown in Figure 4.10b.

4.5 Line Layout under the Periphery Condition

In this last part of the chapter we consider the MLCM problem under the periphery condition. In this variant all lines must be placed outermost in the line orders of their termini. The problem to determine for each line ℓ in $\mathcal{L}$ whether it terminates on the left side or on the right side of its first and last edge is NP-hard [BKPS08]. But once this terminus assignment is fixed, it is known that the remaining MLCM-PA problem of finding the line orders along all edges of the underlying network G can be solved in $O(\mathcal{L}^3 \cdot |E|^{2.5})$ time for any planar graph G [AGM08]. Faster algorithms are known for restricted graph classes [BKPS08, ABKS09], see also Section 4.2. In this section we present a new and significantly faster algorithm that finds, given the terminus assignments, an optimal line layout for any metro graph G in $O(|\mathcal{L}|^2 \cdot |V|)$ time. Moreover, the same algorithm also solves the MLCM-T1 problem, for which the previously best algorithm runs in $O((|E|+|\mathcal{L}|^2) \cdot |E|)$ time [ABKS09].

We first show a simple lemma about the line crossings in an optimal layout for an MLCM-PA instance. A similar result has been obtained by Asquith et al. [AGM08].

Lemma 4.5 *Given a metro graph $(G, \mathcal{L})$ and terminus assignments for all lines in $\mathcal{L}$, all line crossings in an optimal line layout are unavoidable crossings, that is, crossings that are present in any line layout of $(G, \mathcal{L})$.*

Proof. Let ℓ_1 and ℓ_2 be two lines that cross in an optimal line layout along an edge uv. By $P = (w_0, \ldots, w_k)$ we denote the maximal common subpath of ℓ_1 and ℓ_2 that contains uv. First of all note that the crossing along uv is the only crossing of ℓ_1 and ℓ_2 along P; any two subsequent crossings of two lines along a common subpath can be removed by routing the upper line just below the lower line on the edges between the two crossings—this contradicts the optimality of the line layout.

Since there is a single crossing between ℓ_1 and ℓ_2 we can assume that $\ell_1 <^{w_0}_{w_0w_1} \ell_2$ and $\ell_2 <^{w_k}_{w_{k-1}w_k} \ell_1$. These inverted relative orders of ℓ_1 and ℓ_2 in the vertices w_0 and w_k are either enforced by the topology of G because the line orders $<^{w_0}_{w_0w_1}$ and $<^{w_k}_{w_{k-1}w_k}$ must be compatible with w_0 and w_k (if the line continues beyond w_0 or w_k) or by the given terminus assignment (if the line terminates at w_0 or w_k). The only case where the relative order of ℓ_1 and ℓ_2 is not fixed by the compatibility requirements or the terminus assignments is if both lines terminate at the same vertex, say w_0, and are assigned to the same terminus side. In that case, however, they can always be reordered in $<^{w_0}_{w_0w_1}$ such that they reflect their relative order in $<^{w_k}_{w_{k-1}w_k}$ on the opposite end of P and the crossing would disappear. This contradicts the optimality of the layout.

We conclude that the crossing of ℓ_1 and ℓ_2 is unavoidable since the relative order of the two lines at one end of P is inverted at the other end of P due to the given terminus assignments or the compatibility requirements. □

Lemma 4.5 implies that there is a line layout that realizes exactly the unavoidable crossings and, consequently, that any such layout is optimal. The following Algorithm 4.1 first computes all maximal common subpaths of all pairs of lines to determine their relative order. In a second phase all lines are iteratively inserted into the line orders of their edges.

Theorem 4.3 *Given a metro graph $(G, \mathcal{L})$ and terminus assignments for all lines in $\mathcal{L}$, Algorithm 4.1 computes an optimal line layout in $O(|\mathcal{L}| \cdot N) = O(|\mathcal{L}|^2 \cdot |V|)$ time.*

Proof. In Phase 1 of Algorithm 4.1 we compute the table $\text{side}(\cdot, \cdot, \cdot)$, where an entry $\text{side}(\ell_1, \ell_2, uv)$ stores the side to which line ℓ_2 tends with respect to ℓ_1 on edge uv. So if $\text{side}(\ell_1, \ell_2, uv) = \textit{left}$ (*right*), we know that at the end of the maximal common subpath of ℓ_1 and ℓ_2 that contains uv the line ℓ_2 must be placed left (right) of ℓ_1.

In order to compute the set $\Lambda(\ell_1, \ell_2)$ of maximal common subpaths of $\ell_1 = (v_0, \ldots, v_k)$ and ℓ_2 we walk along ℓ_1 and check for each edge v_iv_{i+1} whether ℓ_2 shares that edge with ℓ_1. If this is the case, we either open a new subpath or extend the current subpath. Otherwise we close the current subpath if there is one. We assume that the input $(G, \mathcal{L})$ contains a Boolean edge-line array of size $|E| \times |\mathcal{L}|$ so that we can check whether a line uses an edge in constant time.

For each subpath $\lambda = (v_i, v_{i+1}, \ldots, v_j) \in \Lambda(\ell_1, \ell_2)$ we need to determine whether ℓ_2 *tends* left- or rightward along λ with respect to ℓ_1, that is, whether at the end of λ the line ℓ_2 must be left or right of ℓ_1. There are three cases to consider.

(1) If $v_j = v_k$, that is, ℓ_1 terminates in v_j, and ℓ_2 does not terminate in v_j then ℓ_2 tends leftward (rightward) if ℓ_1 is assigned to a right (left) terminus, respectively.

Algorithm 4.1: MLCM-PA line layout

Input: metro graph $(G, \mathcal{L})$, terminus assignments for all $\ell \in \mathcal{L}$
Output: line orders $<^u_{uv}, <^v_{uv}$ for all edges $uv \in E$

```
/* Phase 1                                                              */
foreach (ℓ1, ℓ2) ∈ L × L, ℓ1 ≠ ℓ2, ℓ1 = (v0, v1, ..., vk) do
    compute set Λ(ℓ1, ℓ2) of all maximal common subpaths of ℓ1 and ℓ2
    foreach (vi, vi+1, ..., vj) ∈ Λ(ℓ1, ℓ2) do
        if ℓ2 leaves ℓ1 towards the left or terminates left of ℓ1 in vj then
            for l = i to j − 1 do
                side(ℓ1, ℓ2, vl vl+1) ← left
        else
            for l = i to j − 1 do
                side(ℓ1, ℓ2, vl vl+1) ← right
/* Phase 2                                                              */
foreach ℓ = (v0, v1, ..., vk) ∈ L do
    for i = 0 to k − 1 do
        insert ℓ into <^{vi}_{vi vi+1}
        insert ℓ into <^{vi+1}_{vi vi+1}
```

(2) If $v_j = v_k$ and ℓ_2 also terminates in v_j, then either ℓ_1 and ℓ_2 are assigned to different terminus sides and ℓ_2 tends to its assigned side, or both are assigned to the same side. In the latter case, ℓ_2 shall stay on the same side of ℓ_1 as in the first vertex v_i of λ. So if ℓ_2 is left of ℓ_1 in $<^{v_i}_{v_i v_{i+1}}$ then it also tends leftward along λ; otherwise it tends rightward.

(3) If $v_j \neq v_k$ then ℓ_2 tends leftward if either ℓ_2 is assigned to terminate on the left in v_j or ℓ_2 continues along an edge $v_j w$ that is left of ℓ_1 in the embedding of the underlying graph G; otherwise ℓ_2 tends rightward.

Since the vertex degree of the underlying metro graph is usually bounded by a small constant (eight in the case of an octilinear layout), side(ℓ_1, ℓ_2, uv) can be computed in constant time in all three cases.

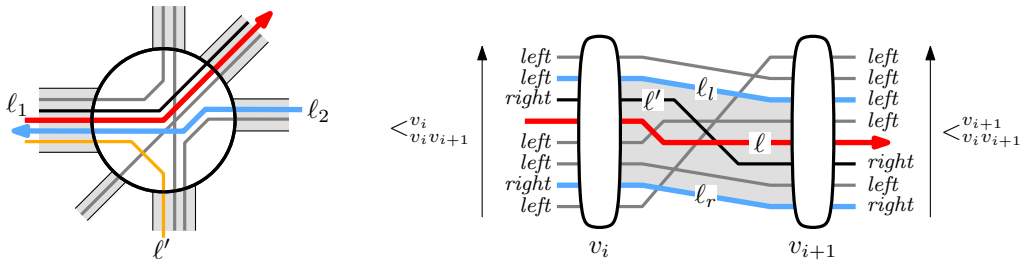
Summarizing the above, Phase 1 takes $O(|\mathcal{L}| \cdot N)$ time and space since we check for each edge of each line if any of the other lines in $\mathcal{L}$ shares the edge; if this is the case we store the corresponding leftward/rightward entry in the table side.

In Phase 2 the actual line layout is computed by iteratively fixing the course of each line. We show the correctness of the algorithm by keeping two invariants during Phase 2.

Invariant 1 There are no invalid intra-vertex crossings, that is, for each vertex u and each edge uv the line order $<^u_{uv}$ is compatible with u.

Invariant 2 All line crossings are unavoidable crossings with respect to the given terminus assignment.

Inserting the first line as the only line into the empty line orders clearly satisfies both invariants. So assume we already have a partial layout that satisfies the invariants and want to insert the next line $\ell = (v_0, v_1, \ldots, v_k)$ into this layout.



(a) Lines ℓ_1 and ℓ_2 are inserted maintaining Invariant 1.

(b) Line ℓ is inserted into $<^{v_{i+1}}_{v_iv_{i+1}}$ maintaining Invariant 2. The values $\text{side}(\ell, \cdot, v_iv_{i+1})$ are indicated for all lines.

Figure 4.11: Insertion of lines into an existing partial line layout.

We start by inserting ℓ into the order $<^{v_0}_{v_0v_1}$. Let's assume ℓ is assigned to a left terminus in v_0 with respect to the first edge v_0v_1 (for a right terminus the insertion is analogous). If ℓ is currently the only line with a left terminus on this edge, we insert ℓ as the last edge into $<^{v_0}_{v_0v_1}$. Otherwise we scan the lines with a left terminus in $<^{v_0}_{v_0v_1}$, starting with the largest element, for the first line ℓ' for which $\text{side}(\ell, \ell', v_0v_1) = \textit{right}$. We insert ℓ immediately left of ℓ'. This first insertion does not create any intra-vertex crossings, so Invariant 1 is clearly satisfied. Furthermore, if there are multiple lines terminating along v_0v_1 on the same side as ℓ then ℓ is inserted exactly between those lines that tend leftward and those lines that tend rightward with respect to ℓ. Hence all those lines are already on the correct side of ℓ and no crossings are created; Invariant 2 is satisfied.

Next, we consider inserting ℓ into the order $<^{v_i}_{v_iv_{i+1}}$ for $i > 0$ such that Invariant 1 is satisfied. If one of the neighboring lines in the previous line order $<^{v_i}_{v_{i-1}v_i}$ also continues along v_iv_{i+1}, then ℓ simply keeps its position next to that line. Since the previous layout did not contain any invalid intra-vertex crossings and ℓ follows a previous line, Invariant 1 is still satisfied. This case is illustrated in Figure 4.11a, where the red line ℓ_1 follows the neighboring black line through the vertex. Otherwise, if ℓ is the only line continuing along v_iv_{i+1}, we scan $<^{v_i}_{v_iv_{i+1}}$, starting with the smallest element, for the first line ℓ' whose previous edge wv_i is left of ℓ in the embedding of G or that terminates in v_i with a left terminus along v_iv_{i+1}. We insert ℓ immediately before ℓ' in $<^{v_i}_{v_iv_{i+1}}$. This is illustrated in Figure 4.11a by the blue line ℓ_2 which is inserted immediately before the yellow line ℓ'. If no line ℓ' is found then ℓ becomes the largest element in $<^{v_i}_{v_iv_{i+1}}$. The chosen position for ℓ ensures that $<^{v_i}_{v_iv_{i+1}}$ remains compatible with v_i and that Invariant 1 is satisfied.

It remains to determine the position of ℓ in the order $<^{v_{i+1}}_{v_iv_{i+1}}$. Figure 4.11b illustrates the situation. We scan the already determined line order $<^{v_i}_{v_iv_{i+1}}$ for the rightmost line ℓ_l left of ℓ for which $\text{side}(\ell, \ell_l, v_iv_{i+1}) = \textit{left}$ and for the leftmost line ℓ_r right of ℓ for which $\text{side}(\ell, \ell_r, v_iv_{i+1}) = \textit{right}$. Note that it is possible that one or both lines ℓ_l and ℓ_r do not exist. If they exist, these two lines ℓ_l and ℓ_r are the closest lines to ℓ that are already on the correct side. Since Invariant 2 holds for the previous partial layout, ℓ_l and ℓ_r do not cross each other along v_iv_{i+1}, that is, $\ell_r <^{v_i}_{v_iv_{i+1}} \ell_l$ and $\ell_r <^{v_{i+1}}_{v_iv_{i+1}} \ell_l$. Obviously, ℓ may not cross either of them and we must insert ℓ between ℓ_r and ℓ_l in $<^{v_{i+1}}_{v_iv_{i+1}}$ (otherwise Invariant 2 will be violated). More precisely, we insert ℓ immediately left of the leftmost line ℓ' in the interval $[\ell_r, \ell_l]$ of $<^{v_{i+1}}_{v_iv_{i+1}}$ for which $\text{side}(\ell, \ell', v_iv_{i+1}) = \textit{right}$, see Figure 4.11b. If ℓ_r (ℓ_l) does not exist we may symbolically assign $\ell_r = -\infty$ ($\ell_l = \infty$) so that the interval $[\ell_r, \ell_l]$ may become unbounded. The position of ℓ is determined as before. If there is no line ℓ' then ℓ becomes the rightmost line in $<^{v_{i+1}}_{v_iv_{i+1}}$.

We claim that in the assigned position ℓ crosses only lines that were to its left and tend to the right or lines that were to its right and tend to the left—crossings that are unavoidable. Assume to the contrary that ℓ crosses a line $\hat{\ell}$ that was to its left and also tends to the left. Since we insert ℓ immediately to the left of ℓ', the two lines $\hat{\ell}$ and ℓ' also cross each other. This is a contradiction to Invariant 2 for the previous partial layout, though, since $\hat{\ell}$ crosses ℓ' from left to right but eventually needs to cross ℓ' again from right to left in order to reach its leftward destination. If there is no line ℓ' then ℓ is the rightmost line in $<_{v_i v_{i+1}}^{v_{i+1}}$ by definition and cannot cross $\hat{\ell}$. Similarly, assume that ℓ crosses a line $\tilde{\ell}$ that was to its right and also tends to the right. Then $\ell_l <_{v_i v_{i+1}}^{v_{i+1}} \tilde{\ell}$ since otherwise we would have placed ℓ left of $\tilde{\ell}$ in the interval $[\ell_r, \ell_l]$. But this means that $\tilde{\ell}$ crosses ℓ_l from right to left, which again violates Invariant 2 for the previous partial layout: there must be a second crossing, where $\tilde{\ell}$ crosses ℓ_l from left to right in order to reach its rightward destination. If $\ell_l = \infty$ we would have placed ℓ left of $\tilde{\ell}$ which is also a contradiction. So Invariant 2 holds for the selected position of ℓ.

Finally, we show that Invariant 1 holds for the position of ℓ in $<_{v_i v_{i+1}}^{v_{i+1}}$. The first case for a potential violation is a line $\hat{\ell}$ with $\text{side}(\ell, \hat{\ell}, v_i v_{i+1}) = \textit{left}$ that is still to the right of ℓ but does not continue further along $v_{i+1}v_{i+2}$. By definition $\hat{\ell}$ can only be right of ℓ if $\hat{\ell} <_{v_i v_{i+1}}^{v_{i+1}} \ell'$. But then Invariant 1 would have been violated before by $\hat{\ell}$ and ℓ'. The other case for a potential violation of Invariant 1 is a line $\tilde{\ell}$ with $\text{side}(\ell, \tilde{\ell}, v_i v_{i+1}) = \textit{right}$ that is still to the left of ℓ but does not continue further along $v_{i+1}v_{i+2}$. By definition this can only be the case if $\ell_l <_{v_i v_{i+1}}^{v_{i+1}} \tilde{\ell}$. But this means that Invariant 1 would have been violated before by $\tilde{\ell}$ and ℓ'.

Since the invariants hold at the end of Algorithm 4.1, we have proven its correctness. The running time of Phase 1 was $O(|\mathcal{L}| \cdot N)$. The running time of Phase 2 is again $O(|\mathcal{L}| \cdot N)$ since there are $2N$ insertion operations, each of which determines a position for the current line by scanning the line orders of size $O(|\mathcal{L}|)$ of the current edge. □

Note that an algorithm for MLCM-PA can also be applied to an instance of MLCM-T1 as the following corollary shows.

Corollary 4.1 *Given an instance of MLCM-T1, that is, a metro graph $(G, \mathcal{L})$ in which all lines terminate in vertices of degree 1, we can use Algorithm 4.1 to compute an optimal line layout in $O(|\mathcal{L}| \cdot N) = O(|\mathcal{L}|^2 \cdot |V|)$ time.*

Proof. In the MLCM-T1 problem we have an implicit periphery condition. Since all termini are located at degree-1 vertices there are no ongoing lines and hence all termini are outermost by definition. We only have to make sure that all terminating lines in a vertex are assigned to the same side of their terminating edge in order to permit all line permutations. Hence Algorithm 4.1 solves MLCM-T1 within the same time bounds. □

4.6 Concluding Remarks

In this chapter we have presented algorithms for two variants of MLCM. In the first part, the unconstrained MLCM problem has been addressed for the fairly restricted case of finding an optimal line layout for a single edge of the underlying network. Our algorithm solves the one-edge problem in quadratic time with respect to the number of lines along that edge.

In the second part, we have given an $O(|\mathcal{L}|^2 \cdot |V|)$-time algorithm that determines an optimal line layout for MLCM-PA, the MLCM problem under the periphery condition with given terminus assignments for all metro lines. The same algorithm also solves the MLCM-T1 problem, where termini are restricted to be located at degree-1 vertices of the underlying graph. This result is highly relevant for real-world metro graph instances. The majority of termini in metro and bus networks is indeed located at degree-1 vertices that often represent remote stations in the suburbs, see Ovenden's book [Ove03]. Frequently, only few terminus stations, for example, downtown bus termini, may have degree more than 1 so that the terminus side assignment in the MLCM-P setting becomes relevant. But even here, this assignment might be specified in the input by the physical location of the corresponding platforms. Otherwise, the ILP formulation of Asquith et al. [AGM08] can be used to determine a crossing-minimal terminus assignment, from which our algorithm for MLCM-PA efficiently computes an optimal line layout.

Open problems. There is a number of interesting open questions that remain in MLCM. On the theoretical side, the complexity status of unconstrained MLCM is still unknown. We have given an efficient algorithm for the very restricted one-edge problem, but it is unknown if the analogous m-edge problem is NP-hard or efficiently solvable, even for $m = 2$. More general MLCM instances consist of trees or, finally, general plane graphs as the underlying graphs. Since the variant MLCM-P is NP-hard it is a natural question to ask for approximations or fixed-parameter algorithms, where, for example, the maximal number of parallel lines per edge is an interesting parameter that is reasonably small in practice.

Another line of generalizing the problem is to relax the restriction that all metro lines are simple paths. In practice, some lines may form cycles or the trains of a single line share a common backbone path only in the central part of the network and branch towards different destinations in the network periphery.

Chapter 5

Dynamic Maps: Morphing Polylines

Interactive dynamic maps are becoming increasingly important in geographic information systems and popular web mapping services such as online atlases or route planning services. Early electronic maps were mainly static images that were scanned from traditional paper maps and that were not designed specifically as screen maps. Due to rapid technological advances most electronic maps today are dynamically created from steadily growing databases. One of the main advantages of dynamic maps and a major reason for their success is that they allow new modes of interaction and personalization that were impossible in static paper maps. Another driving force for interactive maps are mobile devices, such as smartphones, GPS[1] devices, or car navigation systems. These devices become more and more affordable and technically advanced, especially in terms of display resolution and quality, which are critical factors for mobile maps. In the coming years we will see many new map-based applications, but they all rely on efficient methods to display basic geographic features, for example, roads, places, or areas, in a clean and legible way.

In this chapter, we study a problem that arises when displaying linear features like road or river networks in an interactive map that provides a continuous zooming functionality. The complexity of these map features needs to be adapted to the user's target scale in order to avoid overly simple or overly detailed representations. Moreover the shape of the features should change smoothly during the zooming animation. We present a new approach to interpolate between representations of a polyline at two different scales. Our method detects characteristic segments of the polyline and subsequently finds an optimal correspondence between the segments of the two polylines with respect to a suitable morphing distance. In a case study we demonstrate the applicability of our morphing algorithm for real-world road, river, and region boundary data. The chapter is based on joint work with Marc Benkert, Damian Merrick, and Alexander Wolff [NMWB08].

5.1 Introduction

Visualization of geographic information in the form of maps has been established for centuries, see a recent survey on geographic visualization [Nöl07]. Depending on the scale of the map, the level of detail of displayed objects must be adapted in a so-called *generalization* process. Be it done manually or (semi-) automatically, generalization methods usually produce a map at a single target scale. Cartographic generalization is

[1] global positioning system

a well-studied field, and computational approaches have been surveyed, for example, by Weibel and Dutton [WD99] and by Mackaness et al. [MRS07].

In current, often web-based [JW05], geographic information systems users can interactively zoom in and out of the map, ideally at arbitrary scales and with smooth, continuous changes. Current approaches, however, are often characterized by a fixed set of scales or by simply zooming graphically without modifying map objects. In the first case the available scales do not necessarily match the user's desired scale. Moreover, stepping instead of zooming is susceptible to causing loss of context so that the user has to re-orientate. If zooming is done purely graphically, however, the quality of the map at the desired scale often does not match the expected quality since the level of detail is not adapted to the specific scale. To overcome these deficiencies *continuous* generalization methods are needed that aim to generate maps that continuously adapt the degree of generalization to the scale.

This chapter deals with the problem of continuously generalizing linear features such as rivers, roads, or region boundaries between their representations at two scales. Instead of line-simplification methods with a single target scale, we consider interpolating between a source and a target scale in a way that keeps the maps at intermediate scales meaningful. In computer graphics and computational geometry this interpolation process is known as *morphing* [GDCV99]. Of specific interest in our context are morphing algorithms that can deal with the effects of generalization operators such as *exaggeration* and *typification*, which, for example, reduce the number (typification) but increase the size (exaggeration) of road serpentines at the smaller scale.

Contributions. The approach that we present in this chapter can handle the effects of typification and exaggeration. It consists of two steps. In the optional first step our method partitions the input polylines into characteristic segments with roughly uniform curvature (Section 5.3.1). This yields a segmentation of the polylines into straight parts and various bends. Then, based on an appropriate distance function for polylines, we compute an optimal correspondence of the polyline segments at the two input scales in $O(nm)$ time using dynamic programming, where n and m are the respective numbers of characteristic segments per polyline (Sections 5.3.2 and 5.3.3). Unlike general morphing algorithms, this correspondence aims to match semantically equivalent segments of the two polylines. Simple straight-line trajectories are used to define the movement between corresponding points. We have implemented a prototype of the algorithm and demonstrate its applicability in a case study for road network data, the course of a river, and a region boundary (Section 5.4).

5.2 Related Work

Cecconi and Galanda [CG02] studied adaptive zooming for web applications with a focus on the technical implementation. They used the standard Douglas-Peucker line-simplification method [DP73] to generalize linear features. While maps can be produced at arbitrary scales there is no smooth animation of the zooming. A set of continuous generalization operators was presented by van Kreveld [vK01], including two simple algorithms for morphing a polyline to a straight-line segment. Continuous generalization for building ground plans and typification of buildings was described by Sester and Brenner [SB04].

Existing algorithms for the geometric problem of finding an optimal intersection-free

geodesic morphing between two simple, non-intersecting polylines [EHPGM01, Bes02] cannot be applied here because in our setting the pairs of polylines that we want to transform into each other generally do intersect. Surazhsky and Gotsman [SG01a, SG01b] computed trajectories for intersection-free morphings of plane polygonal networks using compatible triangulations. Similarly, Erten et al. [EKP04] gave an algorithm for intersection-free morphing of plane networks using a combination of rigid motion and compatible triangulations. These approaches, however, require a given correspondence between network nodes. In the field of computer graphics, Cohen et al. [CEBY97] matched point pairs of two (or more) parametric freeform curves. They computed an optimal correspondence of the points with respect to a similarity measure based on the tangents of the curves. The algorithm is similar to ours in that it also uses dynamic programming to optimize the matching, but it deals with uniformly sampled points rather than with context-dependent characteristic segments of polylines. Samoilov and Elber [SE98] extended the method of Cohen et al. [CEBY97] by eliminating possible self-intersections during the morphing.

5.3 Model and Algorithm

In our description we focus on the problem of morphing between two polylines, each generalized at a different scale. An algorithmic solution for a pair of polylines can be used to compute a morph between two networks of polylines with identical topology by applying the polyline algorithm for each pair of polylines in the network independently. Note, however, that our algorithm does not take intersections between different polylines into account. Polygonal region boundaries can be handled, too, by cutting the closed curves at an identical point, which then serves as first and last point of the corresponding polylines. The algorithm can be further extended in a straightforward manner to finding a series of morphs across many scales by solving each pair of networks at neighboring scales independently.

The problem of morphing between two polylines is two-fold. Firstly, a correspondence must be found between points on the two lines. Secondly, trajectories that connect pairs of corresponding points must be specified. Here our focus is on the *correspondence problem*. Once we have solved this, we will simply use straight-line trajectories.

In addressing the correspondence problem, our goal is to match parts of each polyline that have the same semantics, for instance, represent the same series of hairpin bends in a road at two levels of detail. We wish to do this in a way that allows the *mental map* to be retained as much as possible. The mental map is the mental image a person builds of a diagram. Retention of the mental map is believed to be important in continuous understanding of animated diagrams; see for example Misue et al. [MELS95]. To retain the mental map, it can be useful to ensure that visual elements change as little as possible during an animation. We therefore wish to minimize the movement of points from one polyline to another. To create a morph with these desired properties, we compute a correspondence between parts of the polylines that is optimal with respect to a distance function defined between polyline segments. This distance function aims to measure the required point movement. Naively, the segments may simply be the individual line segments of the polylines. We can, however, improve the running time of our algorithm by detecting appropriate larger characteristic segments consisting of multiple line segments as described in Section 5.3.1. Whatever segments we use, the algorithm described in Section 5.3.2 computes an optimal correspondence for them.

Formally, we are given two (directed) polylines f and g in the plane $\mathbb{R}^2$. In the *correspondence problem* we need to find two continuous, monotone parameterizations $\alpha : [0,1] \to f$ and $\beta : [0,1] \to g$, such that $\alpha(0)$ and $\beta(0)$ map to the first points of f and g and $\alpha(1)$ and $\beta(1)$ map to the last points, respectively. These two parameterizations induce the correspondence between f and g: for each $u \in [0,1]$ the point $\alpha(u)$ corresponds to $\beta(u)$. The *trajectory problem* asks for a family of trajectories $\sigma(t,u) : [0,1]^2 \to \mathbb{R}^2$ along which $\alpha(u)$ moves to $\beta(u)$, where t is a time point in the interval $[0,1]$. Here we simply use straight-line trajectories, thus connecting $\alpha(u)$ and $\beta(u)$ by shortest possible connections, that is, $\sigma(t,u) = (1-t)\alpha(u) + t\beta(u)$.

One issue with this formulation is that intersections between different parts of a polyline may occur during the morph. We give a heuristic method in Section 5.3.3 that may be implemented to avoid some typical cases of self-intersections, namely if a pair of corresponding segments would intersect; intersections may still occur in some cases between two different parts of the same polyline or between different polylines in a network. The method of Surazhsky and Gotsman [SG01b], who give a solution to the trajectory problem, provides a workaround to this issue by computing more complex but therefore intersection-free trajectories given a solution to the correspondence problem. Since self-intersections did not occur in the examples of our case study we refrained from including their method in our prototype implementation.

5.3.1 Characteristic Points

Before solving the correspondence problem, we need to divide each polyline into subpolylines to be matched up. We do this by locating points on each line that are considered to be characteristic of the line; each of these characteristic points then defines the end of one subpolyline and the start of another.

The simplest approach to locating such points is to assume that every point defining a polyline is characteristic of the line. In this case, we solve the correspondence problem on the set of line segments of the polyline and proceed directly with the algorithm in Section 5.3.2. This method can produce good results as we show in the case study in Section 5.4. Often, however, a large number of points is needed to accurately depict a cartographic feature such as a river or a road, and using all of these points as characteristic points can lead to unnecessarily high running times. To avoid this, we present a method that selects a small subset of characteristic points that still suffice to produce good results in significantly less time.

Previous work on generalization notes the importance of inflection points, bend points, and start- and endpoints in defining the character of a line [PAF95]. We have performed initial experiments with detecting such points automatically [MNWB07], but found that the user had to manually calibrate many parameters in order to obtain reasonable results for a particular polyline. Furthermore, a set of parameters that produced a good solution for one line did not necessarily lead to a good solution for another.

Instead, we detail here an approach that needs only a small set of parameters, and is more robust to changes in the input data. Sezgin [Sez01, Chapter 5] introduces a method for locating feature points of curves in the recognition of hand-drawn sketches. Sezgin tries to model a given polyline with a Bézier curve, and calculates the distance between the Bézier curve and the actual polyline. If this distance is above a certain threshold, the polyline is divided into two parts, for which two new Bézier curves are created. This continues until a set of Bézier curves has been generated each of which fits the polyline

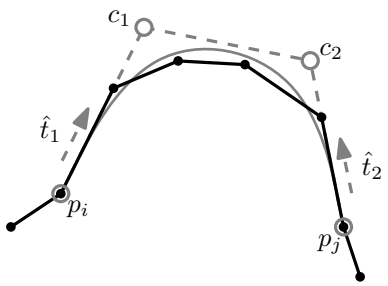


Figure 5.1: A Bézier curve with control points $\langle p_i, c_1, c_2, p_j \rangle$ approximating a polyline between points p_i and p_j.

within the given distance threshold. This process is similar to the classic Douglas-Peucker line simplification method [DP73], but fitting Bézier curves instead of straight lines. The points at which the Bézier curves start and end are considered as the characteristic points of the polyline. This approach tends to produce a set of points that is quite evenly spaced, since the point at which each polyline is divided is arbitrarily chosen as its midpoint.

Our approach also fits Bézier curves to sections of the polyline, but proceeds from the start of the polyline, greedily fitting as many points as possible and starting a new Bézier curve when no more points can be fitted. Given two points p_i and p_j in a polyline $f = \langle p_1, p_2, \ldots p_n \rangle$, we use the same Bézier curve construction as Sezgin [Sez01]: the points p_i and p_j become the first and last control points of the curve, and two intermediate control points are defined as $c_1 = p_i + k\hat{t}_1$ and $c_2 = p_j + k\hat{t}_2$, where k is one third of the length of the polyline $\langle p_i, p_{i+1}, \ldots, p_j \rangle$, $\hat{t}_1$ is the unit vector in the direction from p_i to p_{i+1}, and $\hat{t}_2$ is the unit vector in the direction from p_j to p_{j-1} (see Figure 5.1). Note that the scaling factor k is an empirically determined value that has been reported as working surprisingly well for approximating digitized curves by cubic Bézier curves [Sch88, Sez01].

The algorithm starts by fitting a Bézier curve to points $\langle p_1, p_2, p_3 \rangle$, then adds one point from f at a time until the distance between the curve and the polyline is greater than a given error threshold $\varepsilon > 0$. We use the following simple method to calculate the distance between the curve and the polyline. First, we resample both the polyline and the Bézier curve using the same number x of points. That is, we place x points spaced evenly along the polyline, from one end to the other, and do the same with the Bézier curve. Now we find the maximum distance between any of these x points on the polyline and its corresponding point on the resampled curve. The number x can be set arbitrarily, but it should be greater than the original number of points in the given subpolyline. In our implementation (see Section 5.4), we set x to 300 if an entire polyline is considered, or in the case of a subpolyline, decrease the number proportionally according to the subpolyline's length. It is possible to use more sophisticated error measures here, such as the Fréchet metric (see Section 5.3.3), but we found that this simple measure worked well in practice.

Once the distance error becomes larger than ε, we mark the last considered point p_j as characteristic point, and create a new Bézier curve for $\langle p_j, p_{j+1}, p_{j+2} \rangle$. After we have considered all points in f, the algorithm finishes by marking the last point p_n as characteristic.

The output of our algorithm is the set of characteristic points, which separate the different Bézier curves. Since we use cubic Bézier curves, which are defined by four control points, any interval between two characteristic points should represent at most a single left or right turn or a straight segment of the polyline. More complex shapes cannot be approximated well by cubic Bézier curves.

Since the Bézier curve construction depends on the directions of edges in the original polyline, an unnecessarily high number of characteristic points may be generated in noisy or poorly sampled sections of a polyline. To minimize this, a Gaussian smoothing filter can be applied as a pre-processing step; see Lowe [Low89] for further details on Gaussian filtering and an efficient algorithm. We first resample the polyline using a given number n' of evenly spaced points. We then apply a Gaussian filter with kernel width σ to the resampled line; this essentially moves each point in the line to a weighted average of its neighbors' positions (σ determines the size of the neighborhood that is used).

Applying the Gaussian filter can decrease small variations in direction along the polyline so that each Bézier curve constructed is likely to fit more closely along simple curved sections of the line. This can result in fewer extraneous characteristic points, but can also increase the running time significantly. In Section 5.4, we present results from the Bézier characteristic point detection both with and without the Gaussian filter.

5.3.2 Optimal Correspondence

The previous section described a method for determining a set of characteristic points of a polyline. By subdividing the polyline at the characteristic points we obtain a set of subpolylines (or simply *segments*) that are intended to represent contiguous and homogeneous stretches of the polyline like straight sections or bends with constant curvature.

In this section we assume that the subdivisions of two input polylines f and g into segments are given, for example, as the result of applying the previous characteristic point detection. So let f be divided into n segments $(f_1, \ldots, f_n)$, where each f_i is a subpolyline or a single line segment, and let g be divided into m segments $(g_1, \ldots, g_m)$. We will abbreviate a sequence of segments $f_i, f_{i+1}, \ldots, f_k$ $(i \leq k)$ by $f_{i\ldots k}$.

Now we approach the correspondence problem. Basically there are three possibilities (C1)–(C3) for a correspondence involving a segment f_i:

(C1) f_i is mapped to a single characteristic point (that is, f_i disappears),

(C2) f_i is mapped to a single segment g_k,

(C3) f_i is mapped to a merged sequence of segments $g_{k\ldots(k+r)}$.

Analogously, we denote the three possible types of correspondence involving a segment g_j by (C1′)–(C3′). Clearly, the linear order of the segments along f and g has to be respected by the assignment and each segment can take part in only one of the six possibilities. Mathematically, we model a valid set of such corresponding pairs as what we call a *correspondence relation* $\rho \subseteq \{1, \ldots, 2n{+}1\} \times \{1, \ldots, 2m{+}1\}$, where a segment f_i corresponds to the element $2i \in \{1, \ldots, 2n+1\}$ and the endpoints of f_i correspond to $2i-1$ and $2i+1$. Analogously, segment g_j and its endpoints correspond to $\{2j{-}1, 2j, 2j{+}1\} \subseteq \{1, \ldots, 2m{+}1\}$ such that both ordered sets represent the alternating sequence of characteristic points and segments of f and g, respectively. In order to be valid, ρ has to satisfy the following properties (P1)–(P4) from the perspective of polyline f:

(P1) ρ is *monotone*:
if $(i,k) \in \rho$, $(j,\ell) \in \rho$, and $i < j$ then $k \leq \ell$;

(P2) only contiguous sequences of points and segments can be mapped to an element on the other polyline:
if $(i,k) \in \rho$, $(i,\ell) \in \rho$, and $k < \ell$ then $(i,k') \in \rho$ for all $k < k' < \ell$;

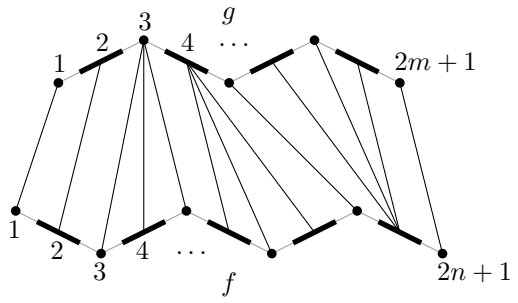


Figure 5.2: Drawing of a correspondence relation ρ between two polylines f and g.

(P3) a merged sequence of elements of one polyline has a unique corresponding element on the other polyline:
if $(i,k) \in \rho$, $(i,\ell) \in \rho$, and $k \neq \ell$ then $(j,k) \notin \rho$ and $(j,\ell) \notin \rho$ for all $j \neq i$;

(P4) all elements are covered by ρ:
for each $i \in \{1,\ldots,2n+1\}$ there is a $j \in \{1,\ldots,2m+1\}$ so that $(i,j) \in \rho$.

Additionally, the symmetric properties from the perspective of g, denoted (P1′)–(P4′), need to be satisfied.

Such a correspondence relation ρ can be seen as a bipartite graph that is a spanning forest in which all trees are (non-trivial) stars. A drawing of such a graph is shown in Figure 5.2, where even-numbered vertices indicate segments and odd-numbered vertices indicate characteristic points. Trees containing a single even-numbered element, that is, a single segment, or trees with an odd-numbered internal vertex represent correspondences of type (C1) or (C1′). Trees containing exactly one even-numbered element on each side mean a one-to-one correspondence of type (C2) or (C2′). Finally, stars with an even-numbered internal vertex and at least two additional even-numbered elements represent correspondences of type (C3) or (C3′).

Now assume that there is a distance function or *morphing distance* δ associated with the morph between two (sub-) polylines. We suggest a morphing distance in the next section, but Algorithm 5.1, which is formulated below, is independent of the choice of the distance. It is based on dynamic programming and computes a minimum-distance correspondence. Algorithm 5.1 recursively fills a table T of size $n \times m$, where entry $T[i,j]$ stores the total distance or *cost* of optimally morphing $f_{1\ldots i}$ to $g_{1\ldots j}$. This total distance is computed as the sum of a previous table entry and the additional distance involving pair (i,j) according to one of the above six types of correspondence. Consequently, we can obtain the cost of an optimal correspondence from $T[n,m]$. By keeping track of optimal subsolutions we can reconstruct the optimal correspondence using backtracking from $T[n,m]$.

The required storage space and running time of filling the $n \times m$ table T in Algorithm 5.1 is $O(nmK)$, which equals $O(nm)$ provided that the integer *look-back parameter* K is constant. This parameter determines the maximum number of polyline segments that can be merged in order to be matched with another segment according to correspondences of types (C3) and (C3′). The final step of reconstructing the actual correspondence is done by backtracking in T and takes linear time. For this analysis we assumed that each distance $\delta(f_i, g_j)$ can be computed in constant time. Depending on which distance function is used, however, the time complexity of computing the required distances needs to be taken into account.

Algorithm 5.1: OptCor

Input: Polylines $f = (f_1, \ldots, f_n)$ and $g = (g_1, \ldots, g_m)$, distance function δ.
Output: Optimal correspondence for f and g.

$T[0,0] = 0$
$T[0,j] = T[0,j-1] + \delta(f_1^{\text{first}}, g_j),\ j = 1 \ldots m$
$T[i,0] = T[i-1,0] + \delta(f_i, g_1^{\text{first}}),\ i = 1 \ldots n$
for $i = 1$ **to** n **do**
  for $j = 1$ **to** m **do**
    $T[i,j] =$

$$\min \begin{cases} T[i-1,j] + \delta(f_i, g_j^{\text{last}}) & \text{type (C1)} \\ T[i,j-1] + \delta(f_i^{\text{last}}, g_j) & \text{type (C1}'\text{)} \\ T[i-1,j-1] + \delta(f_i, g_j) & \text{type (C2)/(C2}'\text{)} \\ T[i-1,j-k] + \delta(f_i, g_{(j-k+1)\ldots j}),\ k = 2, \ldots, K & \text{type (C3)} \\ T[i-k,j-1] + \delta(f_{(i-k+1)\ldots i}, g_j),\ k = 2, \ldots, K & \text{type (C3}'\text{)} \end{cases}$$

    Store pointer to predecessor, i.e., to the table entry that yielded the minimum.
Generate optimal correspondence from $T[n,m]$ using backtracking along pointers.

5.3.3 Distance Functions

Algorithm 5.1 relies on a distance function δ that represents the cost of morphing between two (sub-) polylines. Distance functions for polylines can be defined in many ways. We consider three possible distance functions. Assume that two polylines f' and g' with uniform parameterizations α and β are given. Each point $\alpha(u)$ on f' will move to $\beta(u)$ on g' along the straight-line trajectory $\sigma(t,u) = (1-t)\alpha(u) + t\beta(u)$ of length $|\alpha(u) - \beta(u)|$.

Width

The first distance function considers the longest such segment and is defined as

$$\delta_{\max}(f', g') = \max_{u \in [0,1]} |\alpha(u) - \beta(u)|. \tag{5.1}$$

This value is also known as the *width* of the morph [EHPGM01]. It can be computed in linear time with respect to the complexity of the polylines.

Fréchet

Another well-known metric for polylines is the *Fréchet* metric. It is minimizing the morphing width over all parameterizations of f' and g' and is defined as

$$\delta_F(f', g') = \min_{\substack{\alpha:[0,1]\to f' \\ \beta:[0,1]\to g'}} \delta_{\max}(f', g'), \tag{5.2}$$

where α and β are continuous, increasing functions. The Fréchet distance is often described using the example of a man walking a dog. Then the Fréchet distance is the minimum required length of the dog's leash if the man moves along one polyline and the dog along the other. We used the implementation of the Fréchet metric of van Oostrum and Veltkamp [vOV04]. The running time of this implementation is $O(mn \log^2(mn))$, where m and n denote the complexity of f' and the complexity of g', respectively.

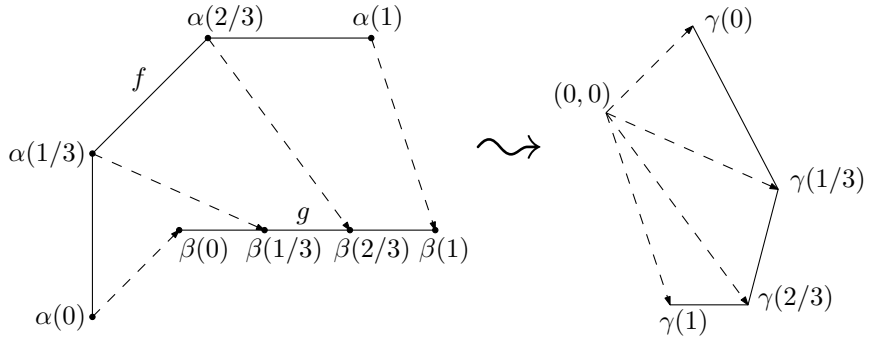


Figure 5.3: The curve γ defined by the parameterizations α of f and β of g.

Integral

Finally, we define a new distance measure that takes into account how far *all* points move during the morph by integrating over the trajectory lengths. This morphing distance is defined as

$$\delta_I(f', g') = \int_0^1 |\alpha(u) - \beta(u)| du \tag{5.3}$$

and can be computed in linear time with respect to the complexity of f' and g'.

Extensions

Relying on a distance function that takes into account only trajectory lengths does not comprehend all aspects that a human expert would consider when trying to match polyline segments optimally. Thus we define further terms that can optionally be added to one of the above base distances.

The first idea is to take into account the length difference of the two subpolylines f' and g'. Subpolylines that have about the same length seem to be more similar than subpolylines of very different lengths. We simply define the cost as the length difference

$$c_{\text{len}}(f', g') = |\, |f'| - |g'|\, |. \tag{5.4}$$

The second idea considers the orientation of the two subpolylines. We want to give preference to matching pairs of subpolylines that are more or less translates. The translation vector of a corresponding pair $\alpha(u)$ and $\beta(u)$ is simply the vector $\beta(u) - \alpha(u)$. These translation vectors themselves define a curve $\gamma(u) = \beta(u) - \alpha(u)$ for $u \in [0, 1]$ which is again a polyline in our case, see Figure 5.3. Thus we can define the length of this polyline γ as a translation cost of the morph between f' and g':

$$c_{\text{tnl}}(f', g') = |\gamma|. \tag{5.5}$$

Note that f' and g' are translates if and only if γ has length zero, that is, all translation vectors are equal; the more the translation vectors vary the larger the translation cost.

The actual distance function δ to be used in Algorithm 5.1 can thus be expressed as a linear combination of a base distance and the above cost terms c_{len} and c_{tnl}. In our implementation the morphing cost $\delta(f_i, g_j)$ of two subpolylines f_i and g_j is further weighted by the ratio $(|f_i| + |g_j|)/(|f| + |g|)$ of the total length of f_i and g_j and the total length of the containing polylines f and g. This accounts for the relative visual weight of the pair (f_i, g_j).

Finally, we wish to avoid self-intersections in the morph. We can do this locally by setting the effective morphing distance to infinity if matching two segments causes a self-intersection in the morph between them. In rare cases, however, intersections between two non-corresponding subpolylines may still occur.

5.4 Case Study

We evaluate our algorithms on three different types of polyline data: (1) a mountain road network in the French Alps, (2) river data, and (3) provincial borders in Germany. We present a detailed case study for the road data and briefly present one example for the course of a river and for a region boundary, respectively.

All experiments were performed on an AMD Athlon XP 2600+ PC with 1.5GB main memory running under SuSE Linux 10.1. The characteristic point detection was implemented in C++ and compiled with gcc 4.2.1; the OptCor algorithm was implemented and tested in Java 1.5.

5.4.1 Road Network Data

We first tested our implementations with a data set of roads in the French Alps from the BD Carto® and the TOP100 series maps produced by the IGN Carto2001 project [LJLH05]. For each road, we used a polyline from BD Carto® at scale 1:50,000, and a generalized version of the same road at scale 1:100,000 from the Carto2001 TOP100 maps. The complete data comprises 382 roads and is shown in Figure 5.4. Details about the network size and the number of characteristic points can be found in Table 5.1. Running times of the Bézier analysis and the OptCor algorithm are given in Table 5.2. Note that in practice the computation of the optimal correspondence is part of the preprocessing of the data and done only once, while the actual morph using straight-line trajectories is computed at interactive speed. Thus even the running times of OptCor in the column *all points* seem acceptable. Due to the size of the network, we will evaluate our method exemplarily for a subnetwork and two single roads that are marked with circles in Figure 5.4. The morph of the complete network as well as animations of further examples can be found on our web site[2].

We start by showing the characteristic points detected for Road 1 in Figure 5.5 and for Road 2 in Figure 5.6. Since the comparison of the results holds at both scales we restrict our description to the scale 1:100,000. Figures 5.5a and 5.6a show all vertices of the polylines as characteristic points. Clearly, these points are very dense where the roads have sharp bends and are more spaced out in parts of less curvature. Thus using all points as characteristic points allows the algorithm to finely adjust the correspondence in parts of high curvature. Applying the Bézier analysis described in Section 5.3.1 reduces the number of characteristic points in particular within dense parts of the roads, as can be seen in the subsequent Figures 5.5b–e and Figures 5.6b–e. For a low threshold value of $\varepsilon = 1$ each bend is still covered by several characteristic points while a higher threshold of $\varepsilon = 25$ leads to finding roughly one characteristic point per bend, just as a human expert would do.

Also note that using Gaussian smoothing prior to the Bézier analysis tends to identify more characteristic points than in the same setting without smoothing. This is perhaps

[2] `http://i11www.iti.uni-karlsruhe.de/morphingmovies`

Figure 5.4: Road network in the French Alps from the BD Carto® map series at scale 1:50,000. The highlighted regions are shown in more detail in Figures 5.5, 5.6, and 5.15.

unexpected, since the smoothing was intended to reduce extraneous characteristic points in noisy sections of a polyline. In fact, the resampling that is performed prior to smoothing creates a much larger number of points that can be chosen as characteristic points. Due to this, it is possible that the Bézier analysis produces a larger number of characteristic points after smoothing, particularly if ε is very low (note how more points are produced when $\varepsilon = 1$ but not when $\varepsilon = 25$).

Note that almost all of the characteristic points marked by the smoothed Bézier analysis lie on line segments between original polyline vertices. This means that the complexity of the polylines is increased artificially by inserting a large number of characteristic points. The unsmoothed cases are more restricted in that only input points can become characteristic points. Finally, for comparison, Figures 5.5f and 5.6f show the results of manually selecting characteristic points located in the peaks of the bends. The number of characteristic points for each example is given in Table 5.1. Due to the relatively small size of these two examples the running times of the Bézier analyses were below 0.01 seconds, see Table 5.2. For the selected subnetwork (see Figure 5.15) and the complete network Table 5.2 shows that the Bézier analysis without Gaussian smoothing remains very fast with up to 0.69 seconds. Smoothing, however, increases the running times to values between 2.4 and 12.22 seconds.

Next, we show the results of the OptCor algorithm for Road 1 and Road 2 using the previously described settings for partitioning the roads. In all our examples the OptCor

network		roads	points	characteristic points $\varepsilon = 1$ s	$\varepsilon = 1$ ns	$\varepsilon = 25$ s	$\varepsilon = 25$ ns
complete	1:50,000	382	13345	9889	6916	2915	2742
	1:100,000	382	10869	7904	5601	2535	2387
subnet	1:50,000	94	1656	1410	930	481	442
	1:100,000	94	1291	1061	739	417	390
Road 1	1:50,000	1	190	112	95	32	32
	1:100,000	1	155	90	72	26	28
Road 2	1:50,000	1	85	68	42	18	16
	1:100,000	1	120	73	53	17	15

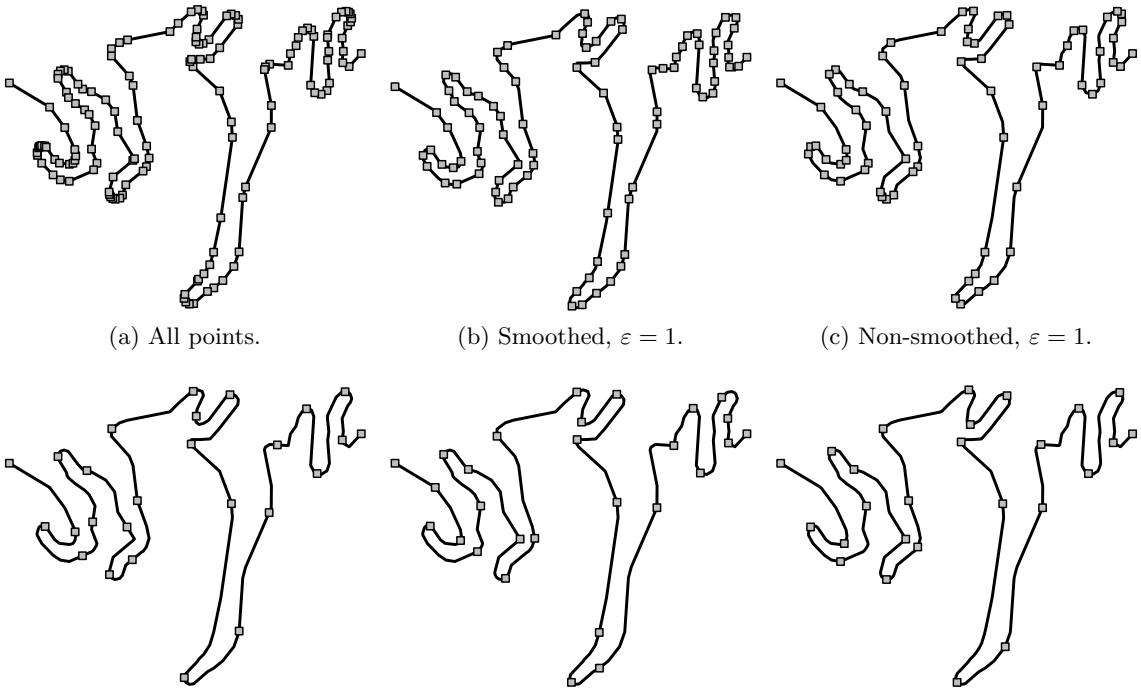
Table 5.1: Sizes of the example networks (**s**: smoothed, **ns**: non-smoothed).

network		all points	characteristic points $\varepsilon = 1$ s	$\varepsilon = 1$ ns	$\varepsilon = 25$ s	$\varepsilon = 25$ ns	manual
complete	Bézier	–	7.72	0.51	12.22	0.69	–
	OptCor	99.84	84.10	43.06	14.83	13.17	–
subnet	Bézier	–	2.40	0.07	3.70	0.13	–
	OptCor	5.88	6.57	2.78	1.29	1.10	–
Road 1	Bézier	–	< 0.01	< 0.01	< 0.01	< 0.01	–
	OptCor	3.15	2.49	1.39	0.62	0.59	0.62
Road 2	Bézier	–	< 0.01	< 0.01	< 0.01	< 0.01	–
	OptCor	1.15	1.18	0.60	0.32	0.27	0.31

Table 5.2: Running times (in seconds) for Bézier analysis and OptCor (**s**: smoothed, **ns**: non-smoothed).

algorithm uses as distance function an equally weighted sum of the integral distance δ_I and extensions c_{len} and c_{tnl} as described in Section 5.3.3. The reason for the first choice was that δ_I turned out to yield better results than both morphing width and Fréchet distance. The look-back parameter was set to $K = 5$. Sequences of snapshots of the final morphs are shown in Figures 5.7 and 5.8 for Road 1, and in Figures 5.9 and 5.10 for Road 2. In each snapshot, previous frames are shown in increasingly light shades of gray to assist perception of the animation. For the purpose of comparison, we show in the same way the result of applying naive linear interpolation, in Figures 5.8c and 5.10c, to produce morphs for Road 1 and Road 2, respectively. Linear interpolation matches each point on one polyline to the point at the same relative distance from the start on the other polyline. We omit the case [smoothed, $\varepsilon = 1$] from the figures here; the quality is comparable to the case [all points].

Four selected pairs of regions have been highlighted and labeled "Region A" through to "Region D" in the snapshots. Close-ups of these regions are given in Figures 5.11–5.14 for a detailed analysis of the morphs. Road 1 is an example where the generalization process applied typification in Region A, that is, the initial set of three bends is transformed into



(a) All points. (b) Smoothed, $\varepsilon = 1$. (c) Non-smoothed, $\varepsilon = 1$.

(d) Smoothed, $\varepsilon = 25$. (e) Non-smoothed, $\varepsilon = 25$. (f) Manual selection.

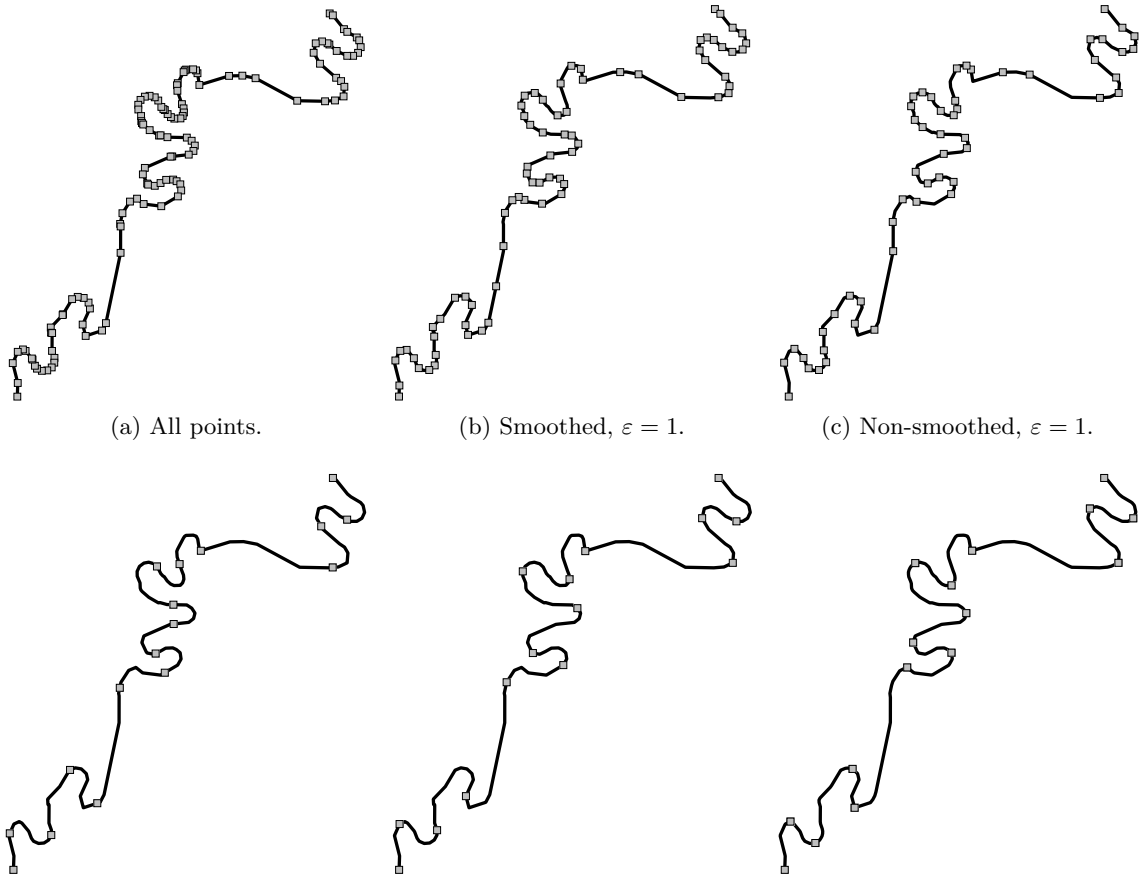
Figure 5.5: Characteristic points for Road 1 (generalized) with Bézier threshold ε.

two bends while maintaining the general pattern of the original set. The generalized version of Road 2 on the other hand exaggerates the bends in Regions C and D, which is also the case in Region B of Road 1.

The morph of Region A shown in Figure 5.11 must deal with the fact that one bend in a series of three bends disappears in the generalized version. While *linear interpolation* (Figure 5.11b) collapses two bends completely in order to recreate one bend, *all points* (Figure 5.11a) merges two bends into a single bend, which we believe is preferable. It is perhaps arguable whether this is the best solution, however. There is obviously a trade-off between obtaining a smooth morph that retains the mental map, and producing the optimal diagram at a fixed scale. If a user stops zooming at an intermediate scale where the merging process is not quite completed it could make sense to continue merging (but keeping the scale) until the representation of the bends is acceptable.

The morph of Region B in Figure 5.12 shows a case where the unsmoothed Bézier analysis with $\varepsilon = 25$ produces far too much excess movement, compared to *all points* (and others). This is due to a rather poor placement of the characteristic points in this case. The number of characteristic points is almost the same in the smoothed and unsmoothed case for $\varepsilon = 25$, see Table 5.1. Still, the smoothed version (Figure 5.8a) leads to a much better morph, which is comparable to the close-up of *all points* in Figure 5.12a. Figure 5.12b also shows another undesirable effect: occasionally, especially for higher values of ε, two neighboring characteristic segments form a sharp "kink" during the morph, which our algorithm currently cannot detect and avoid.

Region C in Figure 5.13 shows a series of bends that are exaggerated slightly in the generalized road. While the *all points* morph (and the other OptCor morphs in Figures 5.9 and 5.10) correctly widen the bends and keep the shape intact, *linear interpolation* collapses



(a) All points. (b) Smoothed, $\varepsilon = 1$. (c) Non-smoothed, $\varepsilon = 1$.

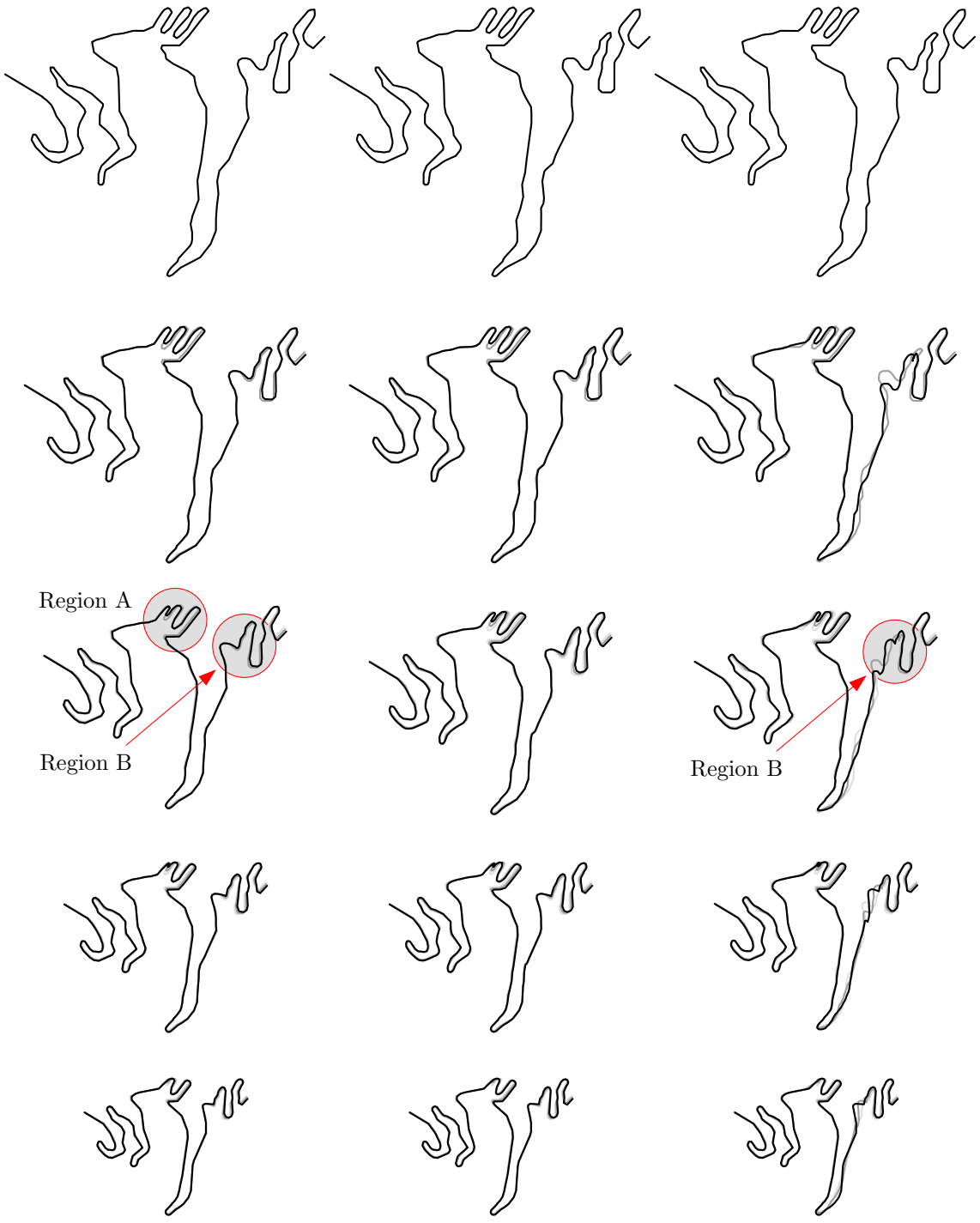
(d) Smoothed, $\varepsilon = 25$. (e) Non-smoothed, $\varepsilon = 25$. (f) Manual selection.

Figure 5.6: Characteristic points for Road 2 (generalized) with Bézier threshold ε.

the bends completely in order to open them up in the inverse direction with a lot of point movement. This shows again the importance of finding a correspondence that minimizes the defined morphing distance.

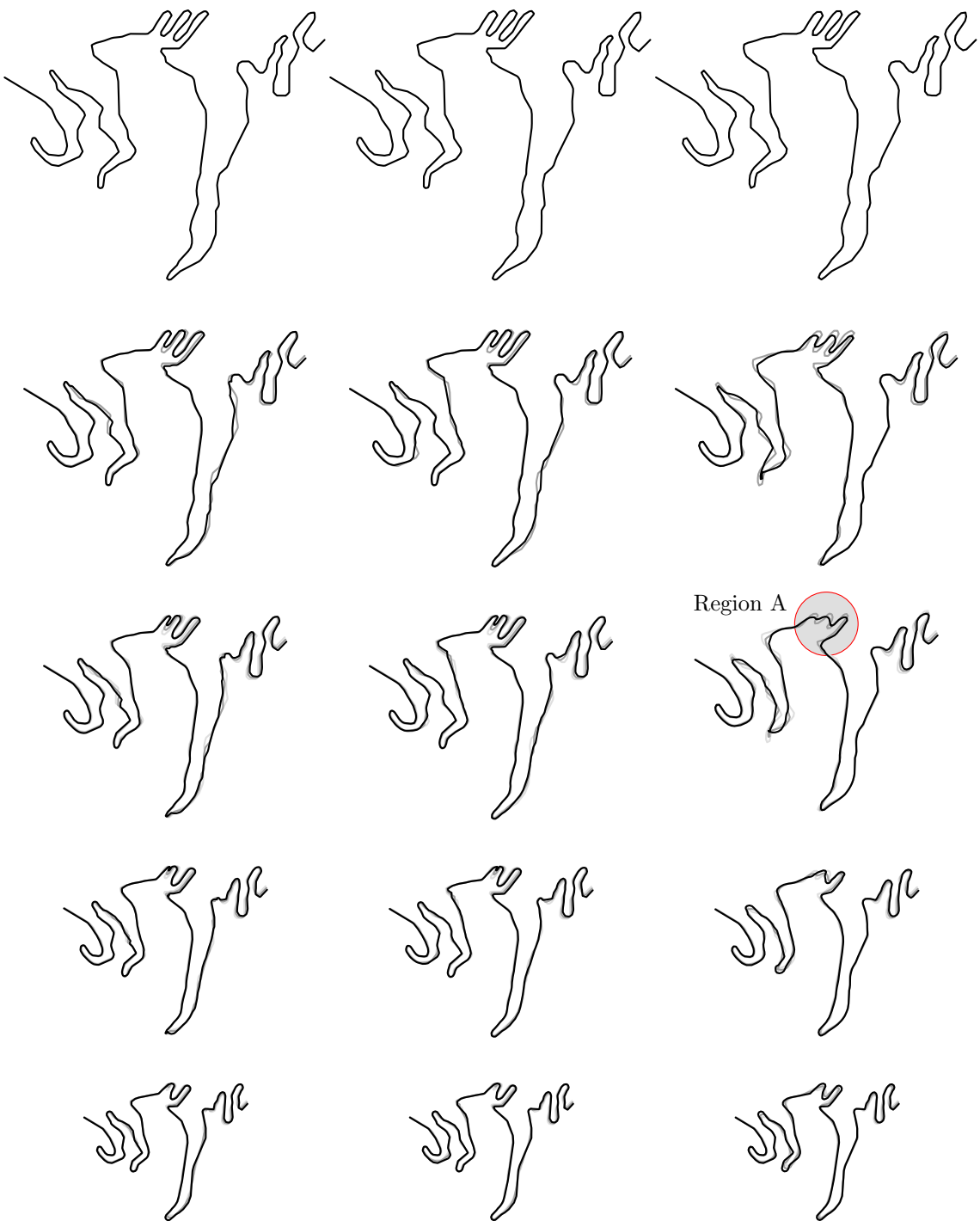
Finally, Region D in Figure 5.14 is another example where two bends are exaggerated. This time the *all points* morph does not increase the bends nicely but rather creates an intermediate "appendix" to the bend at the right-hand side. In contrast, the manually placed points (as well as the setting [smoothed, $\varepsilon = 25$]) lead to a morph without undesired intermediate effects.

We also applied OptCor to the subnetwork highlighted in Figure 5.4, and we show the result using all points as the set of characteristic points in Figure 5.15a. Figure 5.15b shows the linear interpolation of this subnetwork for comparison. Although the networks are drawn in the same size for all three scales it is still difficult to make out the details of such a complex example in this format, but on close inspection one can notice a lot more movement in the linear interpolation compared to the OptCor morph, particularly in the highlighted areas. Like in the examples Road 1 and Road 2, the linear interpolation flattens some bends completely before they reappear (highlighted area in the middle). In the OptCor morph of that area one bend collapses while another one expands. Again the amount of point movement is much less than in the linear interpolation.



(a) All points. (b) Non-smoothed, $\varepsilon = 1$. (c) Non-smoothed, $\varepsilon = 25$.

Figure 5.7: Road 1 morphs generated by OptCor.

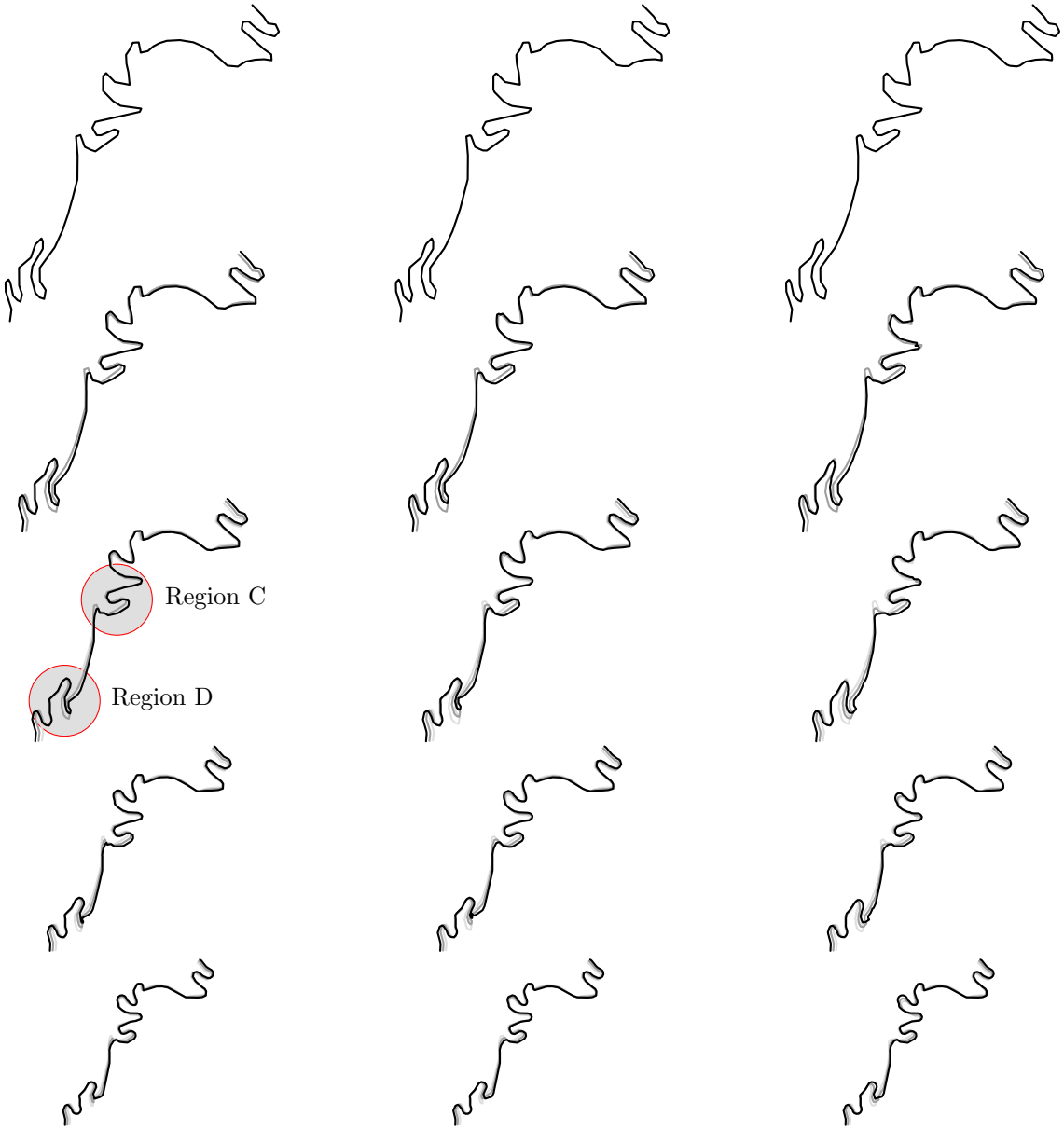


(a) Smoothed, $\varepsilon = 25$.

(b) Manual placement.

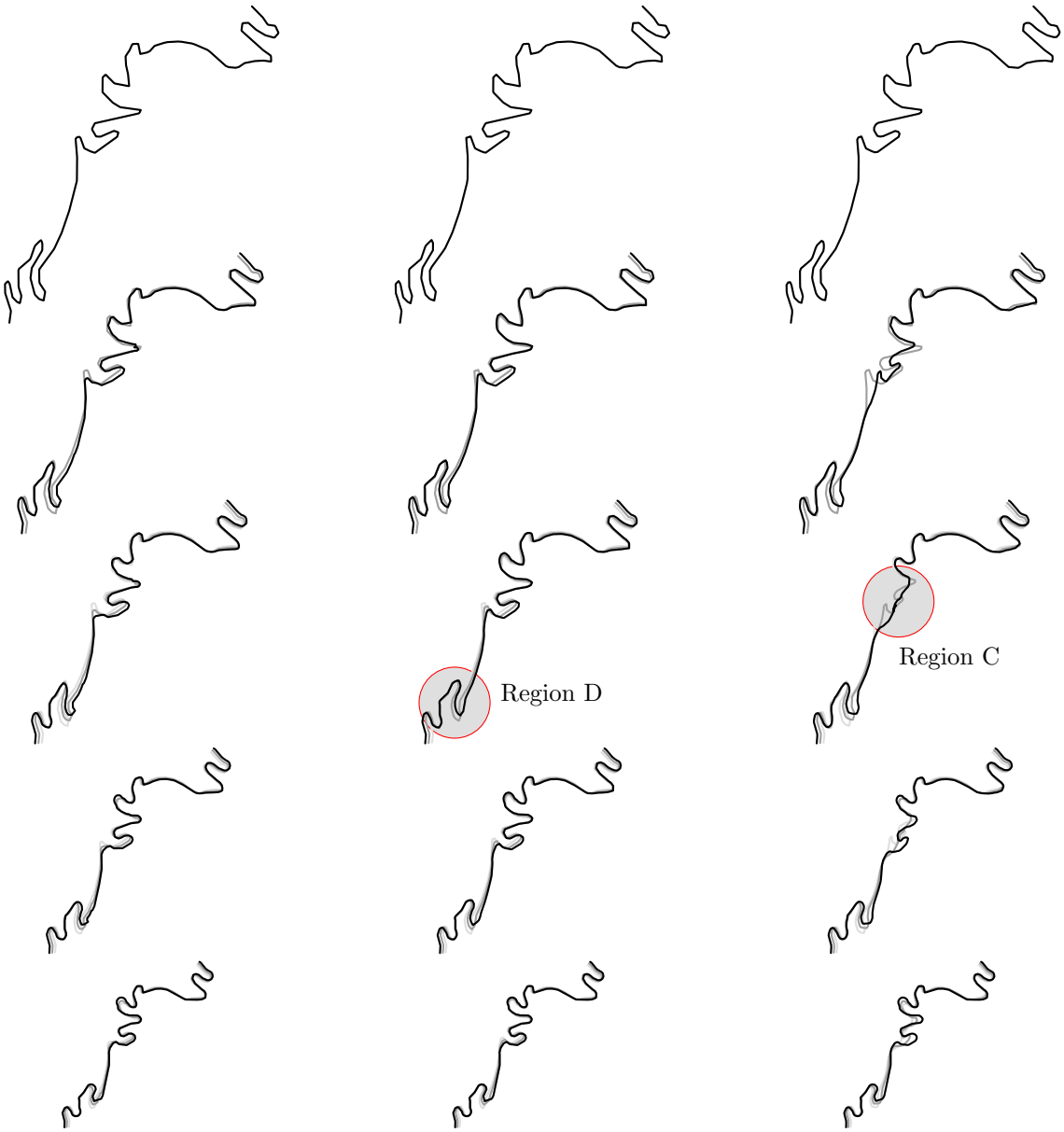
(c) Linear interpolation (without OptCor).

Figure 5.8: Road 1 morphs generated by OptCor.



(a) All points. (b) Non-smoothed, $\varepsilon = 1$. (c) Non-smoothed, $\varepsilon = 25$.

Figure 5.9: Road 2 morphs generated by OptCor.

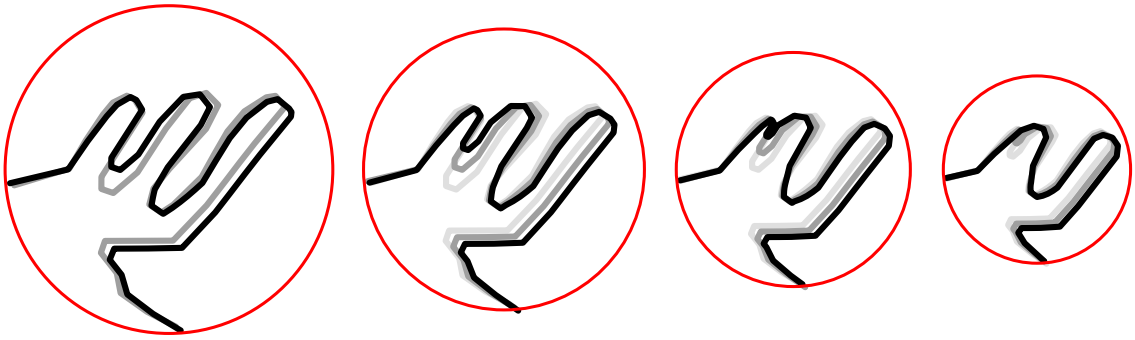


(a) Smoothed, $\varepsilon = 25$.

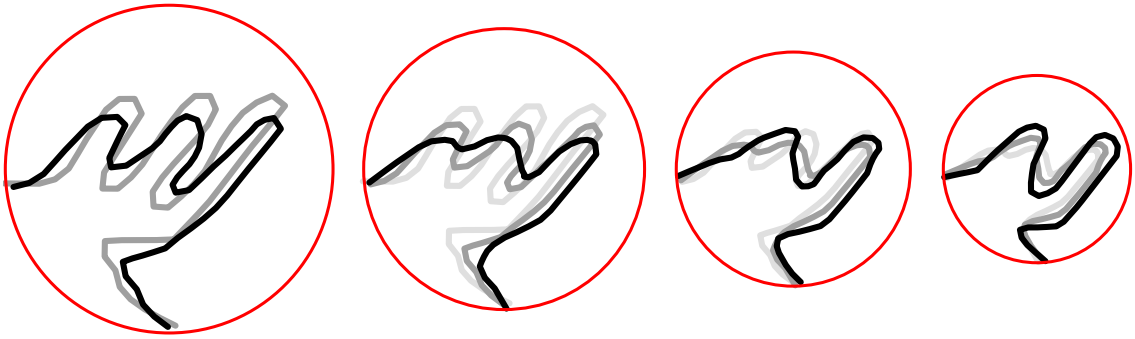
(b) Manual placement.

(c) Linear interpolation (without OptCor).

Figure 5.10: Road 2 morphs generated by OptCor.

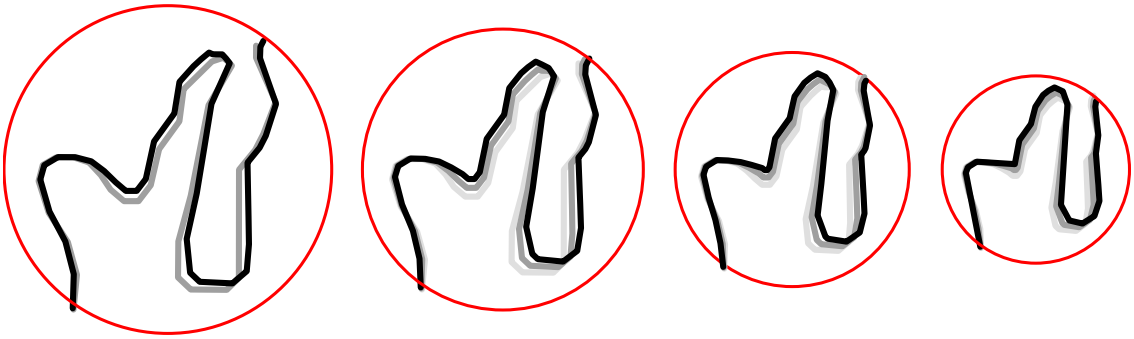


(a) All points.

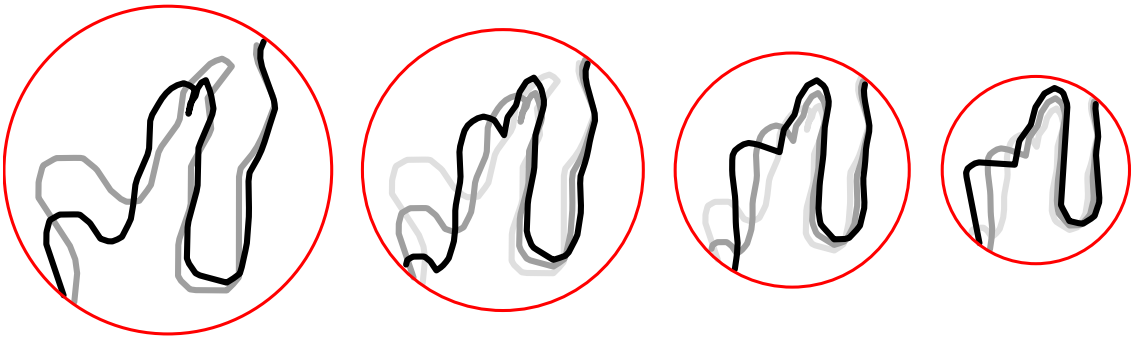


(b) Linear interpolation.

Figure 5.11: Close-up of Region A (Road 1).

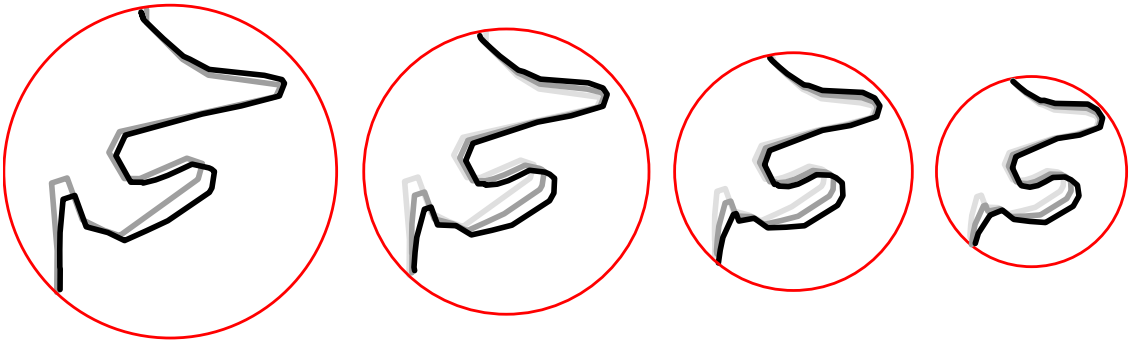


(a) All points.

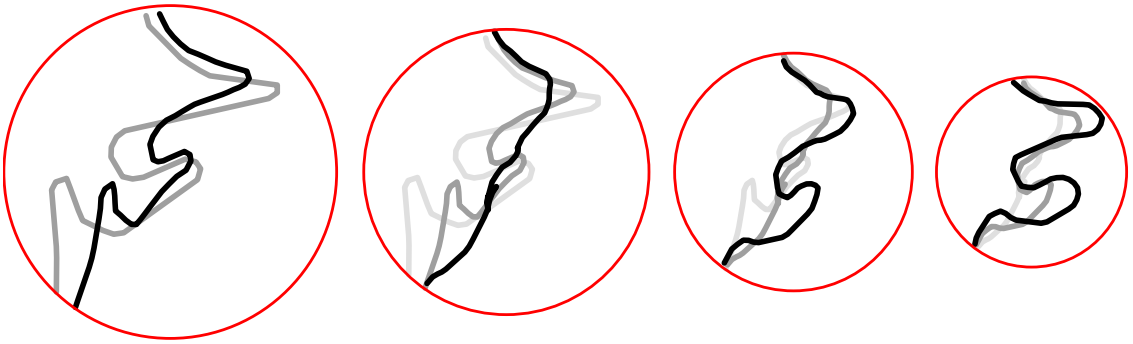


(b) Unsmoothed, $\varepsilon = 25$.

Figure 5.12: Close-up of Region B (Road 1).

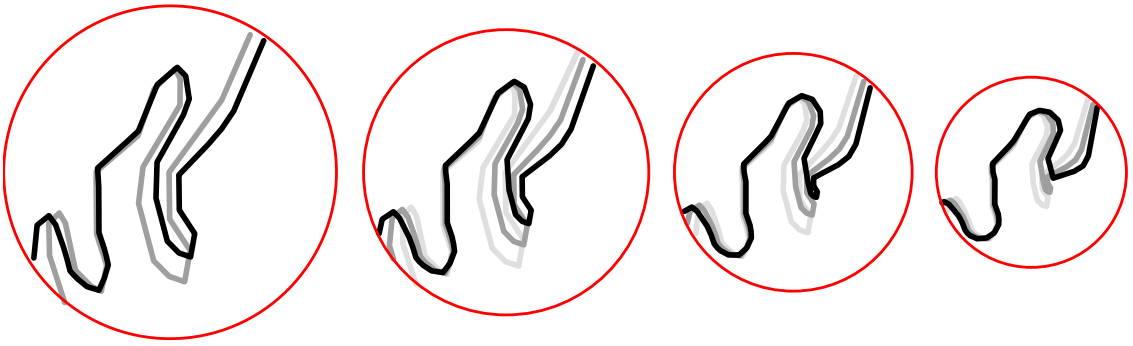


(a) All points.

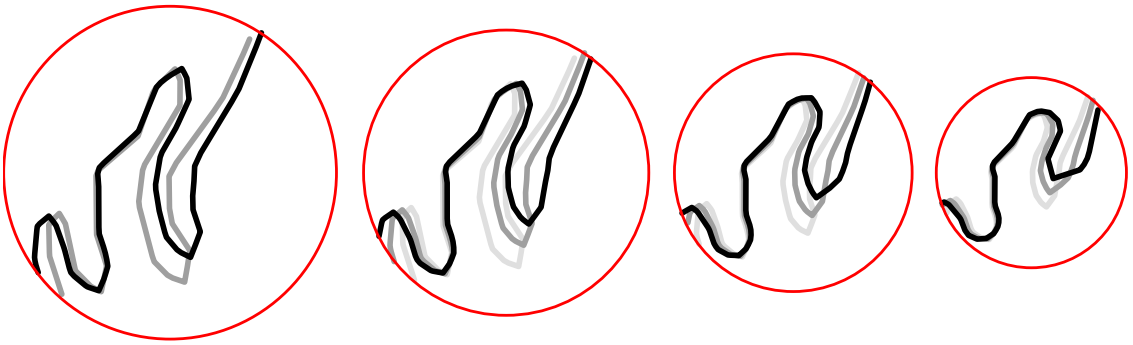


(b) Linear interpolation.

Figure 5.13: Close-up of Region C (Road 2).

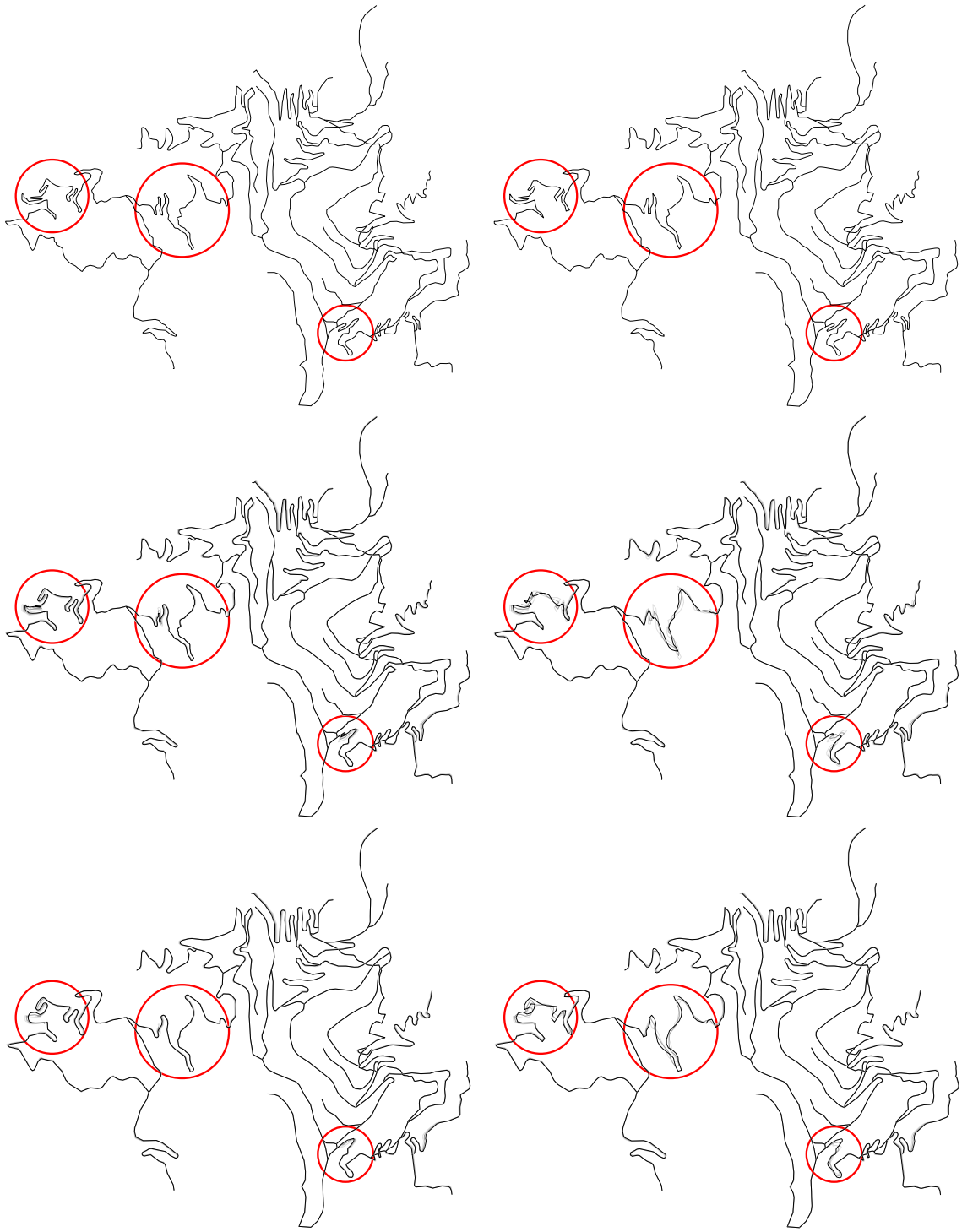


(a) All points.



(b) Manual placement.

Figure 5.14: Close-up of Region D (Road 2).



(a) All points OptCor morph. (b) Linear interpolation.

Figure 5.15: Morph of a subnetwork.

5.4.2 River Data

As another class of polylines we consider a portion of the course of the river Elbe in Brandenburg, Germany. This example stems from the DTK1000 and the Verwaltungsgrenzen 1:2,500,000 data sets (© Bundesamt für Kartographie und Geodäsie, Frankfurt am Main, 2008). The data from the first set has a target scale of 1:1,000,000 and consists of a polyline with 308 points, the data from the second set has a target scale of 1:2,500,000 and uses 210 points.

Figure 5.16 shows a sequence of snapshots from the OptCor morph using all input points as characteristic points in comparison to the linear interpolation of the same data. The representation of the river at the small scale has far less detail than its representation at the large scale, for example, almost all smaller crenulations disappear in the generalized small-scale version. The running time of the OptCor algorithm was 4.8 seconds for this instance. We can draw the same conclusions as previously for the road data: the OptCor morph succeeds in matching semantically equivalent parts of the river at both scales resulting in a smooth morph with almost no excess movement. On the other hand, the naive linear interpolation in Figure 5.16b again erroneously collapses some major bends at intermediate scales before making them reappear at the target scale. This creates unnecessary movement and alters the general shape of the river at intermediate steps, especially in the upper part highlighted in Figure 5.16. The quality of the morphs using characteristic points detected by the Bézier analysis was—similarly to the road network—comparable to the morph using all points; these morphs, too, did not exhibit the rather poor behavior of the linear interpolation.

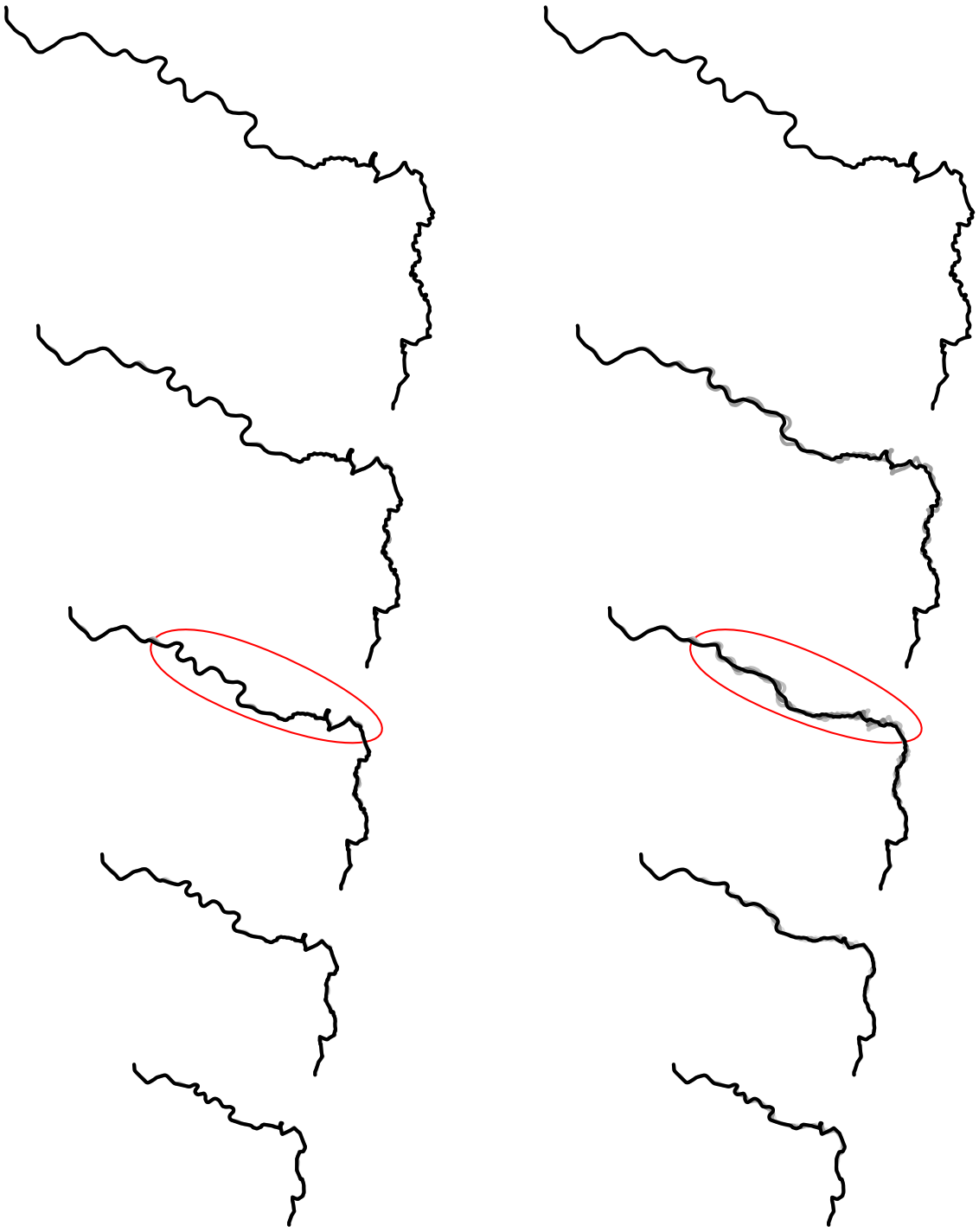
5.4.3 Provincial Border Data

Our final example shows that our method can also deal with polygon data. The source of this example is the boundary of the province Hamburg, Germany from the same data sets as the river data in the previous example. The polygon uses 361 points at the scale 1:1,000,000 and 147 points at the scale 1:2,500,000. Both polygons were transformed into polylines by cutting them at a similar point that served as start- and endpoint of the polylines.

Figure 5.17 shows a sequence of snapshots from the OptCor morph using all input points as characteristic points in comparison to the linear interpolation of the same data. The shape is generalized quite strongly for the smaller scale and the crenulations disappear almost completely. The running time of the OptCor algorithm was 1.8 seconds. While not much movement is visible in the left part of the contour for both morphs, the linear interpolation performs poorly in the upper right part and also the transformation of the lower indentation is noticeably better in the OptCor morph (see the highlighting in Figure 5.17). This shows again that our algorithm is indeed able to retain the viewer's mental map of the contour by finding the optimal correspondence between the characteristic parts of the two input polylines.

5.5 Concluding Remarks

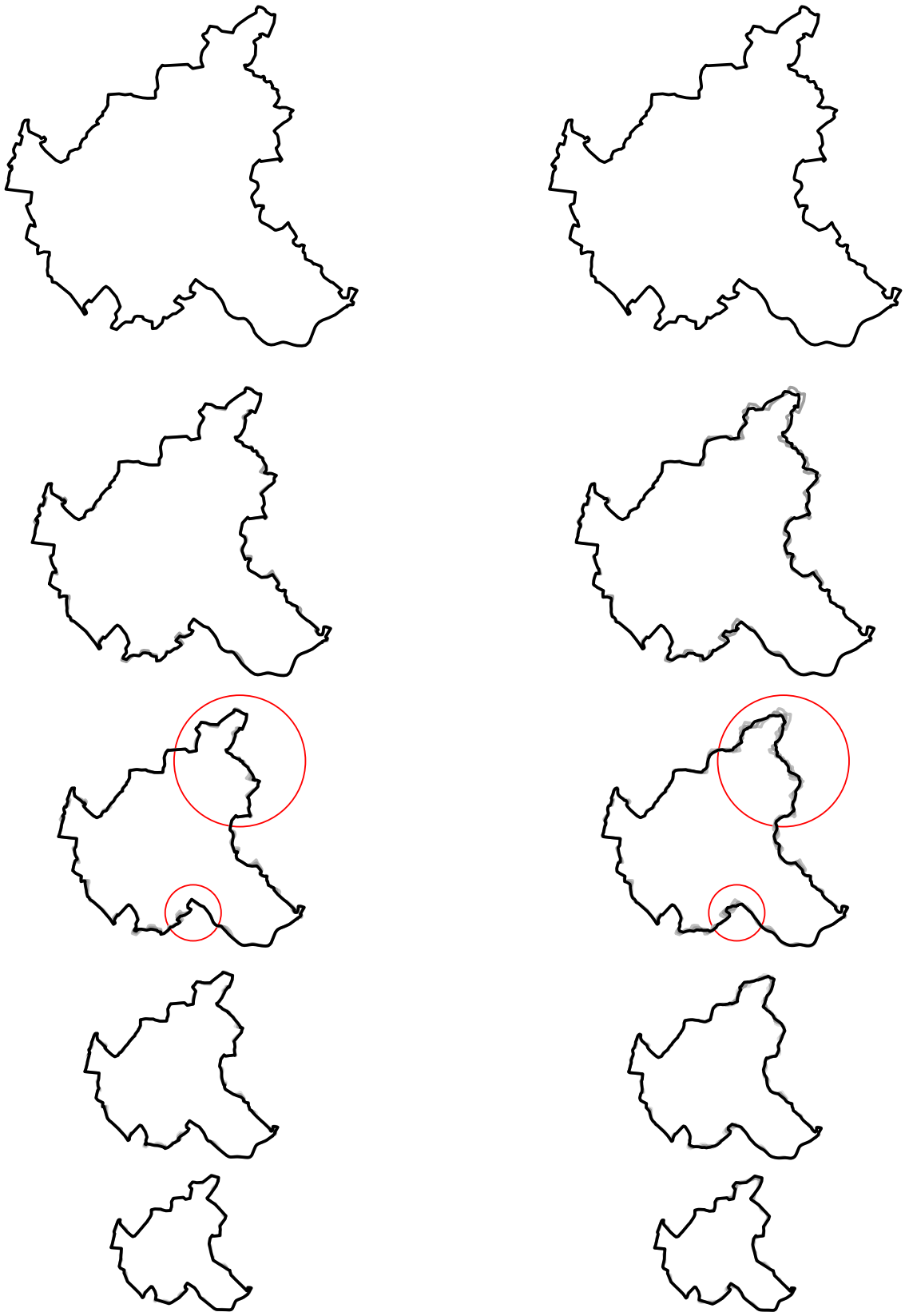
We have presented and evaluated an algorithm to compute an optimal correspondence for two polylines that are partitioned into characteristic segments. We have introduced a



(a) All points OptCor morph.

(b) Linear interpolation.

Figure 5.16: Morph of a portion of the course of the river Elbe.



(a) All points OptCor morph.

(b) Linear interpolation.

Figure 5.17: Morph of the provincial border of Hamburg.

heuristic method to compute these characteristic segments by fitting Bézier curves to the polylines. Our case study indicates that the morphs computed by our method successfully transform the shape of one polyline into another while preserving the mental map of the viewer. These results extend to networks of multiple polylines as well as to polygonal region boundaries. Using the individual line segments of the polylines as the characteristic segments yields good results but at the cost of higher running times. We found that our method for detecting characteristic points reduces the number of characteristic segments while at the same time the quality of the morphs remains generally high. Thus we propose our algorithm as a step towards a more global approach for continuous generalization that also takes the context of further non-linear data layers like point data into account.

Open problems. Apart from the grand challenge of continuously generalizing all features of a dynamic map, our algorithms could be improved in a number of ways. First, ensuring that self-intersections do not occur during a morph can be accomplished by utilizing the algorithm of Surazhsky and Gotsman [SG01b] to compute non-linear trajectories to morph points. Further investigation would be necessary to avoid intersections between different polylines in a network, similar in spirit to static subdivision simplification algorithms that guarantee consistency of the simplification [dBvKS98]. Secondly, it follows from the results in our case study that improved morphs can be obtained in some cases by manually selecting the positions of characteristic points. This asks for a deeper exploration of what constitutes a "good" set of characteristic points. Given a reasonable definition of this, an optimization problem could be formulated for improved characteristic point detection. Finally, even with manually chosen characteristic points, the OptCor algorithm can still produce an occasional "kink" during morphs. It may be possible to design an extension to the distance function that minimizes the occurrence of such kinks.

Chapter 6

Dynamic Maps: Labeling

In this chapter we examine label placement in dynamic maps. Label placement is a classical cartographic problem, but it encounters new and unique issues in the context of interactive dynamic maps. Each map, if static or dynamic, wants to communicate certain spatial information to its readers by displaying point features (cities, mountain summits, etc.), line features (roads, rivers, etc.), and area features (lakes, administrative units, etc.) in the map. Each of these features usually carries a name that uniquely identifies it, at least in some local context (for example, unique street names in a city). Obviously, it is of critical importance to show these names as textual labels in the map such that the reader can quickly identify which label describes which feature. For a traditional static map the labeling problem basically reduces to placing a set of two-dimensional geometric objects (the labels) in the map such that certain constraints are satisfied (for example, no label overlap) and an appropriate objective function is optimized (for example, maximizing the number of labels).

An interactive map that allows the user to zoom and pan continuously still has to satisfy the same constraints as a static map, but it also has additional labeling requirements that arise from the fact that the map is animated during user interaction. For example, labels should not jump, that is, suddenly change their position, nor should they disappear and reappear several times during zooming. In this chapter we define criteria for consistent dynamic map labeling and model the labeling problem as an optimization problem in three dimensions, where scale is the third dimension. Different label shapes are considered and we show that the problem is NP-complete even for quite simple shapes. We present a toolbox of algorithms that yield constant-factor approximations for a number of variants. The chapter is based on joint work with Ken Been, Sheung-Hung Poon, and Alexander Wolff [BNPW08, BNPW09].

6.1 Introduction

Recent years have seen tremendous improvements in web-based, geographic visualization systems that provide continuous zooming and panning functionalities [Nöl07], but relatively little attention has been paid to special issues faced by map labeling in such contexts. In addition to the need for interactive speed, several desiderata for a *consistent dynamic labeling* were identified by Been et al. [BDY06]: labels do not flicker (appear and disappear more than once) or jump (suddenly change position or size) during panning and zooming, and the labeling does not depend on the user's navigation history. Currently available

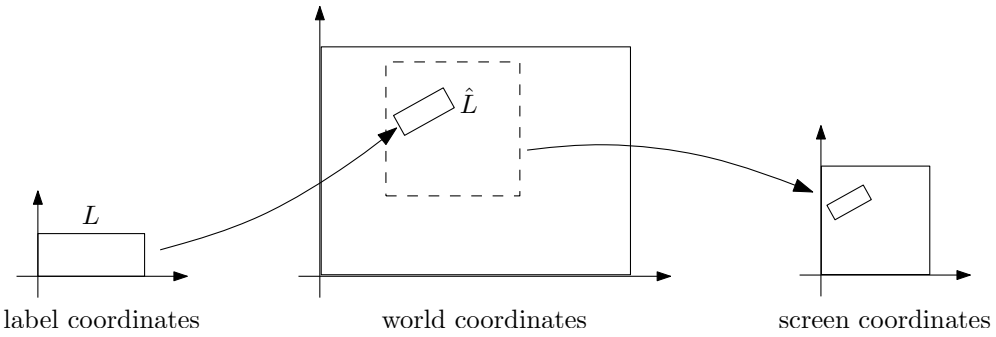


Figure 6.1: Label, world, and screen coordinates.

systems (for example, Google Earth, NASA World Wind, Microsoft Virtual Earth, and KDE Marble) do not satisfy these desiderata and their label placement algorithms may produce more or less unattractive dynamic labelings, at least during animated user interactions.

In this chapter, we determine for each label a single interval of scales for which it is visible. This avoids flickering effects. We restrict our attention to so-called *invariant point placements*, in which a point of reference for each label has a fixed position in the map. This prevents that labels change position during zooming and panning, that is, there is no jumping of labels. Clearly, the scale intervals must be chosen such that no two labels overlap at any scale. We optimize a typical criterion in map labeling, namely we maximize the number of visible labels at each scale. In the following we introduce the model for consistent dynamic map labeling.

Model. We adapt the labeling model of [BDY06] as follows. In *static labeling* the key operations are *selection* and *placement*—select a subset of the labels and place them without overlap. Let each label L be defined in its own label coordinates. A *static placement* of L is the image $\hat{L}$ of L in world coordinates under a transformation composed of translation, rotation, and dilation (see Figure 6.1). A further transformation clips a rectangular region in world coordinates and takes it to the screen, dilated by a factor $1/s$, where we define s as the *scale*. Note that s is the inverse of cartographic scale.

In *dynamic labeling* we select *at each scale* a subset of labels and place them without overlap. To meet the desiderata for consistent dynamic labeling we

(1) define a *dynamic placement* of L to be a function that assigns a static placement $\hat{L}(s)$ to each scale $s \geq 0$;

(2) require that each dynamic placement be continuous with scale;

(3) define *dynamic selection* to be a Boolean function of scale;

(4) require that each label L_i, $1 \leq i \leq n$, be selected precisely on a single interval of scales, $[a_i, A_i]$, which is called the *active range* of L_i.

Note that the active range may also be empty, that is, the label is never shown, which is different from the case $a_i = A_i$, where the label is shown at precisely one scale.

We define *extended world coordinates* by adding a scale dimension to world coordinates. Then we can think of dynamic placement as mapping a label L into extended world coordinates such that the cross section of the image of L at scale s is the static placement $\hat{L}(s)$. Let $S_{\max}$ be a universal maximum scale for all labels. We define the *available range* of L_i

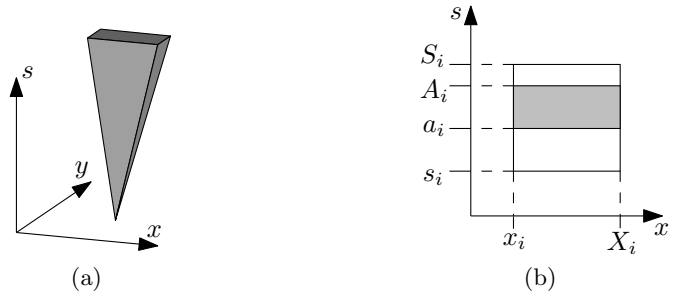


Figure 6.2: (a) A dynamic placement of a 2d-label is a solid in extended world coordinates. Here: with invariant point placement and proportional dilation. (b) A 1d-label with constant dilation, available range $[s_i, S_i]$, and active range $[a_i, A_i]$.

to be an interval of scales, $[s_i, S_i] \subseteq [0, S_{\max}]$, in which label L_i "wants" to be selected. For example, street labels are available at smaller scales and country labels at larger scales. Note that scales $s < 1$ actually mean that world coordinates are magnified on screen; for maps in cartography we will usually have $s \gg 1$. For the active range of a label L_i we require $[a_i, A_i] \subseteq [s_i, S_i]$. Let $E_i = \bigcup_{s\in[s_i,S_i]} \hat{L}_i(s)$, that is, E_i is the union of all static placements of L_i within its available range, and let $T_i = \bigcup_{s\in[a_i,A_i]} \hat{L}_i(s)$ be the restriction of E_i to the active range $[a_i, A_i]$. Since dynamic placement is continuous with scale, E_i is a solid defined by sweeping the label shape along a continuous curve that is monotonic in scale, with the rotation and dilation factors at each scale given by continuous functions. An example is given in Figure 6.2a. We call E_i the *extrusion* of L_i and T_i its *truncated extrusion*.

The extrusion shapes are determined by the label shape and the translation, rotation and dilation functions that compose the dynamic placement. We restrict our attention to certain classes of extrusions, which are simple yet relevant in applications. Our 2d-labels are rectangular (for example, bounding boxes of textual labels); for theoretical interest, we also consider 1d-labels, which are horizontal segments $[x_i, X_i]$ in 1d world coordinates—see Figure 6.2b. In this chapter we consider dynamic placements that are so-called *axis-aligned invariant point placements*: the rotation component is constant and maps L to an axis-aligned rectangle at each scale; the translation component is constant and maps a particular reference point of the label always to the same location in world coordinates, that is, the label never "slides"; the dilation component is a linear function $D^L(s) = bs + c$, for which we consider three classes.

- If $b = 0$ and $c > 0$, that is, $D^L(s) = c$, then label size is fixed in world coordinates and inversely proportional to scale on screen. Thus labels shrink at the same rate as the geographic features when zooming out and grow when zooming in. The solid is then a "straight" extrusion, as in Figure 6.2b.

- If $b > 0$ and $c = 0$, that is, $D^L(s) = bs$, then L has constant size on screen and size proportional to scale in world coordinates. The solid is then a label-shaped cone with apex at $s = 0$ as in Figure 6.2a. With invariant point placements, the cone contains the vertical line through its apex. The cone might be symmetric to that line (for example, for labeling a region) or it might be slanted and have this line as a vertical corner edge (for example, for labeling a point).

- If $D^L(s) = bs + c$ for constants $b > 0$ and $c \neq 0$, then label size is disproportionate to scale in world coordinates and on screen. For $c < 0$, the label size increases on screen when zooming out and decreases when zooming in. For $c > 0$, the label size on screen decreases when zooming out and increases when zooming in, albeit not as fast as the geographic features—unlike the dilation function $D^L(s) = c$. The solid in this case is a portion of a label-shaped cone with apex at $-c/b$.

Objective. Let $\mathcal{E}$ denote the set of all extrusions, and assume we are given an available range for each. For a set $\mathcal{T}$ of truncated extrusions, define $H(\mathcal{T}) = \sum_{i=1}^{n}(A_i - a_i)$ to be its *total active range height*. This is the same as integrating over all scales a function that counts the number of labels selected at scale s. The *(general) active range optimization (ARO)* problem is to choose the active ranges so as to maximize H, subject to the constraint that no two truncated extrusions overlap. This is the dynamic analogue of placing the maximum number of labels without overlap in the static case. We concentrate solely on maximizing H since alternative optimization criteria like maximizing the minimum active range height do not make much sense in practice—for example, if there is a label with a very small available range. We call any set of active ranges that correspond to non-overlapping truncated extrusions a *solution*. It is of theoretical and practical interest to also consider a version of the problem in which all labels are available at all scales and a label is never deselected when zooming in—that is, $[s_i, S_i] = [0, S_{\max}]$ and $a_i = 0$ for all i. We call this variant of ARO *simple*.

Related problems. The following packing problem is a special case of 1d-ARO: given a set $\{E_1, \ldots, E_n\}$ of axis-aligned unit squares as extrusions, find pairwise disjoint truncated extrusions $T_1 \subseteq E_1, \ldots, T_n \subseteq E_n$ of maximum total active range height. We do not know the complexity of this packing problem, but we give an efficient (2/3)-approximation algorithm (Theorem 6.7) and we do show that the problem is hard, if the squares have two different sizes (Theorem 6.1). Clearly, the problem is closely related to geometric maximum independent set problems, that is, maximum independent set problems in intersection graphs of geometric objects. Such problems are usually NP-hard and admit—if the geometric representation is given—polynomial-time approximation schemes, for example in the case of axis-aligned unit squares [HM85], even if the size restriction is dropped [EJS05]. In those problems, however, one can choose only to either put or not to put a complete object into a solution, whereas in ARO, one can put an arbitrarily small fraction of an object into the solution. We insist only that each object contributes at most one connected component to the solution and that such a component must fill the whole width of the object.

The above-mentioned special case of 1d-ARO can also be viewed as a scheduling problem with geometric constraints. To see this, use *line stabbing* [AvKS98]: stab the unit squares with vertical lines of distance greater 1 such that each square is stabbed and each line stabs a square. Now each stabbing line corresponds to a machine, each square corresponds to a job that is run on the corresponding machine, and the y-axis is the time axis. Each machine executes at most one job at a time, and a job can be started at most once. While a job is being processed, it blocks other jobs if the corresponding squares intersect.

Finally, there is some similarity to dynamic storage allocation, where one gets requests for blocks of contiguous bytes of memory. Each request has a start and an end time. In this problem, the x-axis is the time axis and the positive integers on the y-axis correspond to memory cells. A request corresponds to a rectangle of fixed size that can slide vertically. Each request must be placed, and the aim is to minimize the amount of memory that has

to be allocated, that is, the smallest y-coordinate occupied by a rectangle. Buchsbaum et al. [BKK+04] give a $(2+\varepsilon)$-approximation algorithm for this problem and polynomial-time approximation schemes for a number of special cases.

Previous work. Map labeling has been identified as an important application area by the Computational Geometry Impact Task Force [Cc99], and has been the focus of extensive algorithmic investigation [WS96]. The vast majority of research on this topic covers *static* labeling. A typical goal is to select and place labels without overlap while optimizing an objective function, which might be simply the number of labels [AvKS98, vKSW99], or it might incorporate multiple cartographic criteria [CMS95]. There are many variations possible, and most have been shown to be NP-hard [FW91, KR92, vKSW99].

For *dynamic* labeling, Petzold et al. [PGP03, PPH99] described an algorithm that uses a preprocessing phase to generate a reactive conflict graph, in which each label is a vertex and each edge corresponds to a potential conflict between two labels. The edges are augmented by an interval of scales in which the conflict occurs. In the interaction phase, the conflict graph is queried with a clipping region and a target scale and a static conflict graph is derived containing all labels in the selected region and all edges for which the conflict interval contains the query scale. From this graph a valid labeling is computed using static labeling heuristics. The method of Poon and Shin [PS05] builds a hierarchy of precomputed solutions; interpolation between these solutions produces a solution for any scale. Neither of these approaches satisfies the consistency desiderata. In addition to introducing consistency for dynamic map labeling, Been et al. [BDY06] showed that simple 2d-ARO is NP-complete for arbitrary star-shaped labels, and implemented a simple heuristic solution.

Contributions. We investigate the complexity of ARO in Section 6.2. We prove that general 1d-ARO with constant dilation is NP-complete, even if all extrusions are squares, and that simple 2d-ARO with proportional dilation is NP-complete, even if all extrusions are congruent square cones. Both proofs are by reduction from Planar3-Sat, the latter using 3d gadgets. We present an algorithmic study of ARO in Section 6.3 by developing an algorithmic toolbox containing both new techniques and new applications of known techniques to solve several variants of ARO problems. One of our algorithms is exact, the others yield approximations. Table 6.1 summarizes our results. Note that all our results for 2d-ARO with congruent square cones can be generalized to congruent and non-rotated rectangular cones by dilating the input space in x- or y-direction. Analogously, the result for arbitrary square cones can be generalized to similar, non-rotated rectangular cones.

6.2 Complexity

In this section, we prove that two variants of ARO are NP-complete, namely general 1d-ARO with constant dilation (see Section 6.2.1) and simple 2d-ARO with proportional dilation (see Section 6.2.3). We also show that we can reduce a variant of 1d-ARO with constant dilation to 1d-ARO with proportional dilation, see Section 6.2.2.

Both hardness proofs reduce from the NP-complete problem Planar3-Sat [Lic82] that has been introduced in Section 2.3.

d	extrusion shape	– technique	ARO	dilation	NPC	approximation	running time $O(\cdot)$	reference
1	triangles	– dynamic programming	simple	bs	no	optimal	n^3	Thm. 6.4
	unit squares	– line stabbing	general	c	?	2/3	$n \log n$	Thm. 6.7
	unit-height rectangles	– MIS		c	yes	1/3	$n \log n$	Thm. 6.5
	unit-width rectangles	– line stabbing		c	yes	1/2	$n \log n$	Thm. 6.6
	rectangles	– divide & conquer		c	yes	$1/\log n$	$n \log n$	Thm. 6.8
	segments of congruent triangles			bs	?	1/2	$(k+n) \log n$	Thm. 6.11
	congruent trapezoids			$bs + c$	?	1/2	$k \log n + n \log^2 n$	Thm. 6.9
2	congruent square cones		simple	bs	yes	1/4	$(k+n) \log^2 n$	Cor. 6.1
				bs	yes	$1/4 - \varepsilon$	$n/\varepsilon \cdot \log(n/\varepsilon) \log n$	Thm. 6.14
	arbitrary square cones			bs	yes	1/24	$n \log^3 n$	Thm. 6.13
	segments of congruent square cones		general	bs	yes	1/4	$(k+n) \log^2 n$	Thm. 6.12
	congruent square frusta			$bs + c$	yes	$1/(4W)$	$(k+n)n^2$	Thm. 6.10

Table 6.1: Results attained in this paper, where d is the dimension of the ARO problems, NPC means NP-complete, k is the number of pairwise intersections between (side edges/faces of) extrusions, $\varepsilon > 0$, and W is the width ratio of top over bottom side of the congruent square frusta.

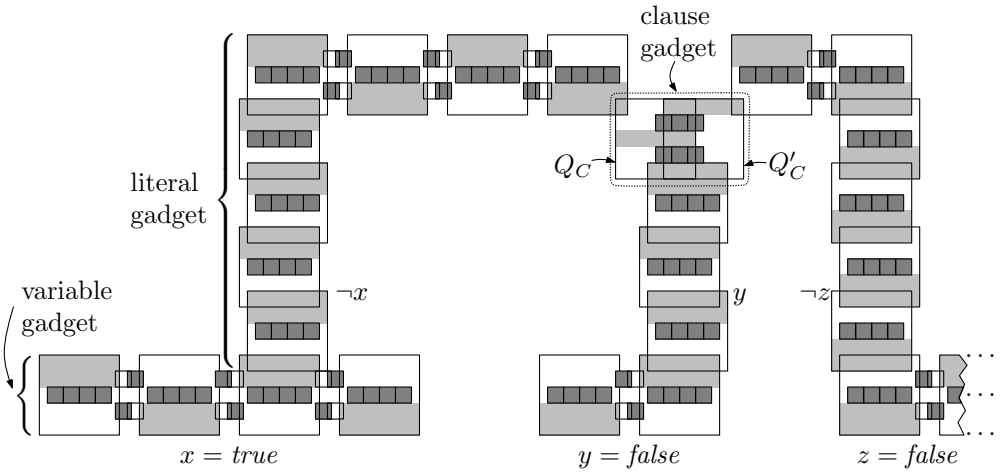


Figure 6.3: The gadgets of our reduction for the clause $C = (\neg x \vee y \vee \neg z)$.

6.2.1 General 1d-ARO with Constant Dilation

Theorem 6.1 *General 1d-ARO with constant dilation is NP-complete; that is, given a set $\mathcal{E} = \{E_1, \ldots, E_n\}$ of axis-aligned rectangular extrusions in the plane and a real number $K > 0$, it is NP-complete to decide whether there is a set of pairwise disjoint truncated extrusions $\mathcal{T} = \{T_1, \ldots, T_n\}$ and $H(\mathcal{T}) \geq K$, where $T_i \subseteq E_i$ for $i = 1, \ldots, n$. The problem remains NP-complete when restricted to instances where all extrusions are squares and each has one of two sizes.*

Proof. For membership in $\mathcal{NP}$, decompose each E_i into $O(n)$ horizontal strips determined by the lines $\{s = s_i, s = S_i \mid 1 \leq i \leq n\}$. It is not hard to see that there is an optimal solution that corresponds to a union of such strips. So we can guess a subset of the strips and then check in polynomial time whether (a) all strips from the same square are consecutive, (b) no two strips overlap, and (c) their total height is at least K.

To show hardness, let φ be an instance of Planar3-Sat, that is, a planar 3-Sat formula with n variables and m clauses. We construct a set $\mathcal{E}_\varphi$ of squares as illustrated in Figure 6.3 and fix a threshold $K > 0$ such that $H(\mathcal{S}(\mathcal{E}_\varphi)) \geq K$ for an optimal solution $\mathcal{S}(\mathcal{E}_\varphi)$ if and only if φ is satisfiable.

The squares in $\mathcal{E}_\varphi$ have side length 1 or 5. We refer to a square of side length j as a j-square. A *regular* 5-square is a 5-square that contains a vertically and horizontally centered chain of four interior-disjoint 1-squares. Thus the 5-square can contribute at most two units to H if the 1-squares contribute one unit each. This is a total of six units for the group of five squares, which is more than if the 5-square contributed five units and the 1-squares zero. Geometrically, the contribution of a 5-square is a (5×2)-rectangle that can either appear above or below the chain of 1-squares. We say that the 5-square is in upper or lower *state*, which gives us a means to encode Boolean values. The contributing part of each square is shaded in Figure 6.3.

Each square in $\mathcal{E}_\varphi$ belongs to a variable gadget, a literal gadget, or a clause gadget. These gadgets correspond one-to-one to the n variables, $3m$ literals, and m clauses of φ.

Variable gadgets. The gadget of a variable x consists of a horizontal chain of n_x regular 5-squares, where n_x is proportional to the number of times that x appears in φ. Two adjacent 5-squares are joined by two *connectors*, that is, pairs of 1-squares as depicted in

Figure 6.3. Each 1-square of a connector overlaps both the adjacent 5-square and the other 1-square in the connector. Since the two 1-squares of a connector overlap, a connector can contribute at most one unit in total. Moreover, if two adjacent 5-squares were in the same state, one of the connectors would contribute zero. Therefore, in any optimal solution for a variable gadget the states of adjacent 5-squares alternate. We let x being *true* correspond to the leftmost 5-square of the gadget being in upper state. Summing up yields that the total contribution of the gadget of x to H is $6n_x + 2(n_x - 1) = 8n_x - 2$ units.

Literal gadgets. A clause of φ consists of three literals. A literal gadget connects a variable gadget to a clause gadget, implementing one of the three legs of the aforementioned comb. The gadget of a literal λ consists of a vertical part and, if λ corresponds to the left or right leg of a comb, a horizontal part. A vertical part consists of a chain of regular 5-squares where consecutive squares overlap by one unit. A horizontal part is identical to a variable gadget, see Figure 6.3. The last square of the vertical part is the first square of the horizontal part.

Number the 5-squares in each variable gadget from left to right. If λ is negated, the first 5-square of the gadget of λ overlaps the top of an odd-numbered (or the bottom of an even-numbered) 5-square of the corresponding variable gadget by one unit. Otherwise parity flips; see the positions of the gadgets of literals $\neg x$ and y in Figure 6.3.

Note that the vertical part of a literal gadget contributes maximally to H (that is, with six units per regular 5-square) if all 5-squares are in the same state as the intersected 5-square of the variable gadget.

If the gadget of a literal λ has a horizontal part, then the states of the 5-squares in that horizontal part alternate as in a variable gadget. We insist that any horizontal part consists of an even number of 5-squares. Thus the state of the final 5-square of the gadget of λ is opposite to that of the first 5-square. The literal gadget can be seen as a mechanical construction that transmits pressure from the variable gadget into the clause gadget: if the 5-square where a literal gadget is attached to its variable gadget from the top is in upper state (corresponding to *false*) then the active ranges of all 5-squares of the literal gadget are "pushed" towards the clause gadget; otherwise, if the literal is *true*, there is no pressure towards the clause.

For a literal λ, let m_λ be the number of regular 5-squares, and let m'_λ be the number of connectors in the gadget of λ. Then λ contributes at most $6m_\lambda + m'_\lambda$ units to H.

Clause gadgets. The final square of each literal gadget connects to a clause gadget. A clause gadget consists of two overlapping 5-squares Q_C and Q'_C, containing six 1-squares as depicted in Figure 6.3. If the six 1-squares contribute one unit each, the two 5-squares can also contribute at most one unit each. This is by construction also the maximum contribution of a clause gadget. Let $Q \in \{Q_C, Q'_C\}$. Note that there are three scale intervals in which Q might contribute one unit to H, and that Q overlaps with the final 5-square of two of the three legs corresponding to literals in C. Further note that the literal gadgets have enough slack to make the final 5-squares overlap Q as shown in Figure 6.3.

Assume that the literal legs contribute maximally to H. Then, if the two literal legs overlapping Q evaluate to *false*, only the middle unit-height strip of Q can contribute (one unit) to H. But since Q_C and Q'_C overlap, their two middle strips *together* can contribute at most one unit. Thus, if all three literals evaluate to *false*, the clause gadget (Q_C, Q'_C and the six 1-squares in their union) contributes 7 units in total. On the other hand, if at least one literal in C evaluates to *true*, both Q_C and Q'_C can contribute one unit, and the clause gadget contributes 8 units. This is the case in Figure 6.3, where Q_C contributes its

middle strip and Q'_C contributes its top strip.

Reduction. The variable, literal, and clause gadgets form the set $\mathcal{E}_\varphi$ of extrusions representing φ. It remains to fix the threshold K such that $H(\mathcal{S}(\mathcal{E}_\varphi)) \geq K$ if and only if φ is satisfiable. Note that a variable gadget contributes maximally to H if and only if the states of its 5-squares alternate, that is, it correctly encodes either the value *true* or the value *false*. Similarly, a literal leg contributes maximally to H if and only if it correctly transfers the value of the literal (or *false*) to its final 5-square. Finally, a clause gadget contributes maximally to H if and only if at least one of its literal legs encode the value *true*. Thus φ is satisfiable if and only if all gadgets in our construction contribute maximally to H.

Let $K = (8\sum_{v\in\mathrm{Var}(\varphi)} n_v - 2) + (\sum_{\lambda\in\mathrm{Lit}(\varphi)} 6m_\lambda + m'_\lambda) + 8m$, where $\mathrm{Var}(\varphi)$ and $\mathrm{Lit}(\varphi)$ denote the sets of variables and literals in φ, respectively. The summands of K correspond to the maximum contributions of all variable gadgets, literal gadgets, and clause gadgets. By the above observation a total active range height of at least K can be achieved if and only if φ is satisfiable.

The set $\mathcal{E}_\varphi$ consists of $O(m^2)$ squares since the variable-clause graph of φ can be drawn on a grid of size $O(m^2)$ [KR92]. The positions of all squares can be encoded in space quadratic in the length of an encoding of φ. The reduction can be performed in polynomial time. □

6.2.2 General 1d-ARO with Proportional Dilation

We have not succeeded in showing that general 1d-ARO is hard in any of the following cases:

(i) All extrusions are unit squares,

(ii) all extrusions are unit-width rectangles,

(iii) all extrusions are trapezoidal segments of congruent triangles whose apexes lie on the x-axis.

By specialization, problem (ii) is at least as hard as problem (i). We now show that problem (iii) is at least as hard as problem (ii).

Theorem 6.2 *There is a polynomial-time reduction from general 1d-ARO with constant dilation for unit-width rectangles (problem (ii)) to general 1d-ARO with proportional dilation (problem (iii)).*

Proof. We take an instance I of general 1d-ARO with constant dilation, that is, a set of n unit-width rectangles, and transform it in a three-step process into an instance I' of general 1d-ARO with proportional dilation, that is, a set of n trapezoidal segments of congruent triangles whose apexes lie on the x-axis, such that $H(I) = H(I')$.

Let B be the height of the tallest rectangle in I, let Δx be the minimum non-zero horizontal distance between any two vertical rectangle edges in I, and let $\varepsilon = \Delta x/n$.

1. First we go through I from right to left. For each vertical line ℓ that contains vertical edges of $t > 1$ rectangles $E_1, \ldots, E_t$ (numbered lexicographically with respect to, say, upper left corners), we add a horizontal "gap" of width $(t-1)\varepsilon$ that moves all rectangles that lie completely to the right of ℓ by this amount horizontally to the right. Then for $i = 1, \ldots, t$ we move E_i to the right by $(i-1)\varepsilon$. Note that at the end of this step,

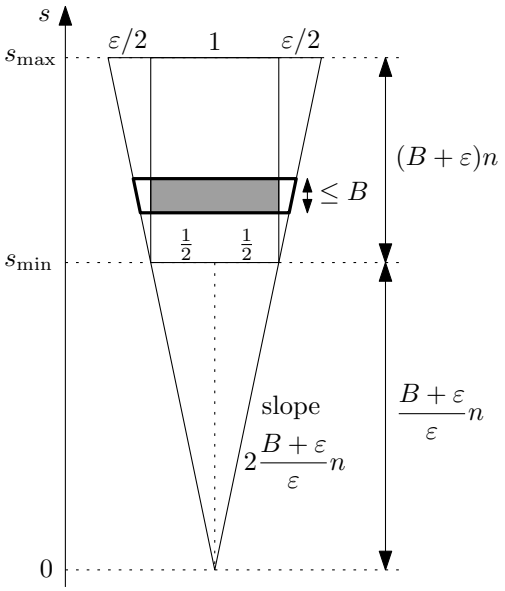


Figure 6.4: Transforming a rectangle (shaded) into a trapezoidal segment (bold boundary) of a skinny triangle.

a) every pair of vertical rectangle edges has distance at least ε, and

b) every rectangle has moved to the right by a total amount of at most $(n-1)\varepsilon < \Delta x$. Thus no two vertical rectangle edges have changed their x-order.

2. Next we vertically "compress" the resulting instance to an instance of height at most $(B+\varepsilon)n$ by replacing each maximal empty horizontal strip by one that has height ε. We move the instance into the horizontal strip of height $(B+\varepsilon)n$ bounded by the lines $s = s_{\min} := (B+\varepsilon)n \cdot h$ and $s = s_{\max} := (B+\varepsilon)n \cdot (h+1)$, where $h = 1/\varepsilon$.

3. Finally we transform the rectangles into trapezoidal segments of skinny congruent triangles—without changing the conflict graph for any scale s. Each trapezoidal segment inherits the available range from the corresponding rectangle. We set the slope of the triangle side edges to $\pm 2(B+\varepsilon)n/\varepsilon$ and place the triangle apexes on the x-axis, vertically below the rectangle centers, see Figure 6.4. By the choice of the horizontal strip in the previous step the trapezoidal segment of the skinny triangle induced by the intersection with that strip has a bottom edge of unit width. Note that at the end of this step

   c) every trapezoidal segment contains the corresponding rectangle and is symmetric to the vertical line through the rectangle center,

   d) every trapezoidal segment has width at most $1+\varepsilon$ (this is the width of the intersection of the line $s = s_{\max}$ with any of the skinny triangles),

   e) two trapezoidal segments intersect if and only if the corresponding rectangles intersect, even more: the height of the intersection of two trapezoidal segments is the same as the height of the intersection of the corresponding rectangles.

Observations (a) to (e) yield the correctness of our reduction. It is clearly polynomial. □

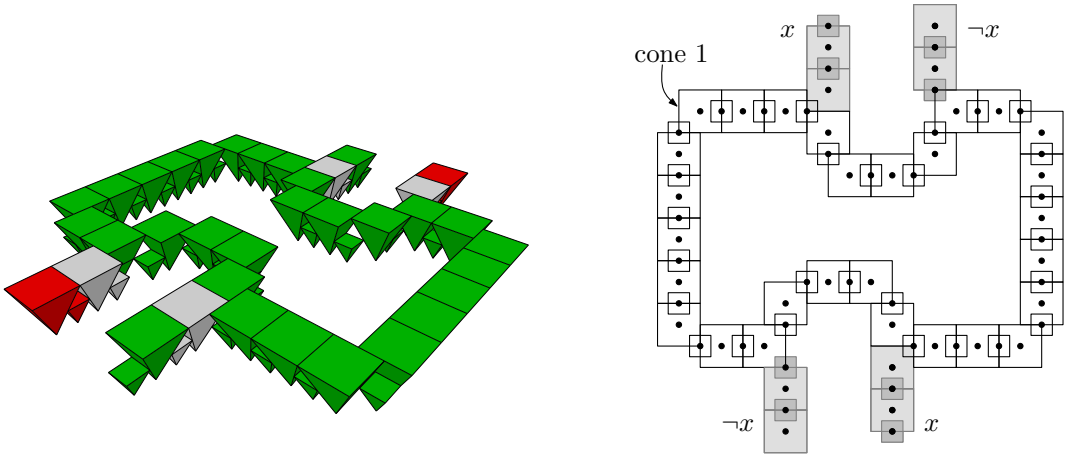
6.2.3 Simple 2d-ARO with Proportional Dilation

Going from the one-dimensional problem to the two-dimensional problem, even *simple* ARO, where each available range is $[0, S_{\max}]$, becomes hard.

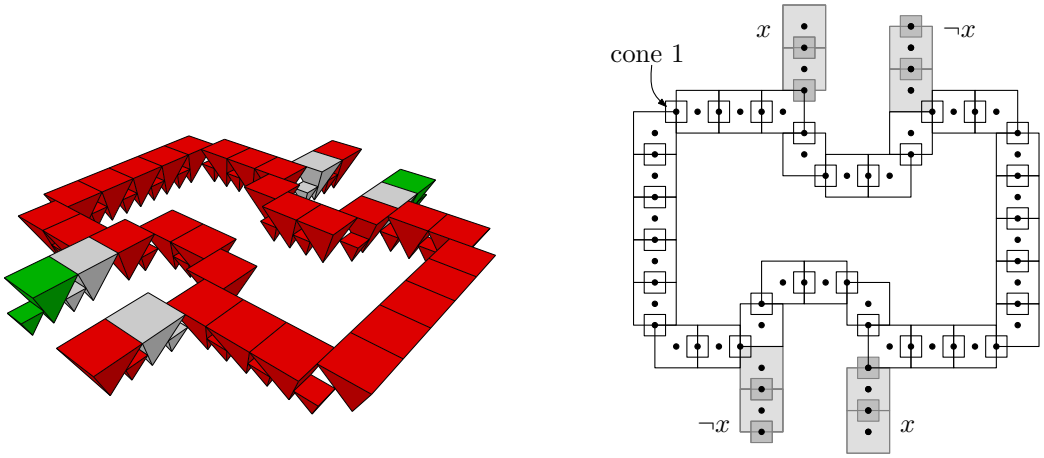
Theorem 6.3 *Simple 2d-ARO with proportional dilation is NP-complete; that is, given a set $\mathcal{E} = \{E_1, \ldots, E_n\}$ of axis-aligned rectangular cones and a real number $K > 0$, it is NP-complete to decide whether there is a set of pairwise disjoint truncated extrusions $\mathcal{T} = \{T_1, \ldots, T_n\}$ and $H(\mathcal{T}) \geq K$, where $T_i \subseteq E_i$ for $i = 1, \ldots, n$. The problem remains NP-complete when restricted to instances where all extrusions are congruent square cones.*

Proof. To see membership in $\mathcal{NP}$, note that in any optimal solution $\mathcal{S}$ each cone either reaches $S_{\max}$ or touches another cone. Thus $\mathcal{S}$ can be constructed by guessing an order in which the cones are greedily "filled" as far as possible from bottom to top.

To show hardness, let φ be a planar 3-SAT formula with n variables and m clauses. We construct a set $\mathcal{E}_\varphi$ of congruent unit square cones and show that there is a threshold $K > 0$ such that $H(\mathcal{S}(\mathcal{E}_\varphi)) \geq K$ for an optimal solution $\mathcal{S}(\mathcal{E}_\varphi)$ if and only if φ is satisfiable.

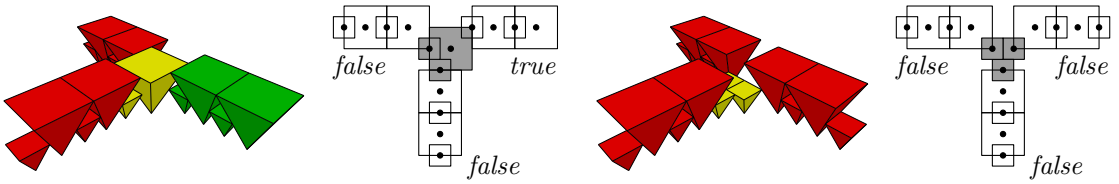


(a) Variable gadget for $x = \mathit{true}$.



(b) Variable gadget for $x = \mathit{false}$.

Figure 6.5: 3d-models and 2d-projections of a variable gadget and partial literal gadgets. Cones of gadgets that carry the value *true* (*false*) are green (red).



(a) Clause gadget, where one literal is *true*.

(b) Clause gadget, where all literals are *false*.

Figure 6.6: 3d-models and 2d-projections of a clause gadget (three yellow/gray cones in the center) with partial literal gadgets. Cones of literal gadgets carrying the value *true* are green, cones of literal gadgets carrying the value *false* are red.

Similar to the 1d-case, we construct variable gadgets and three-legged combs connecting them to clause gadgets. Figure 6.5 shows a variable gadget and Figure 6.6 shows a clause gadget. Variable gadgets and comb legs consist of chains of cones whose apexes lie on a half-integer grid, that is, a grid whose grid point coordinates are integer multiples of $1/2$. Cones whose apexes have L^∞-distance $1/2$ intersect at scale $S_{\max}/2$ while those with L^∞-distance 1 intersect at scale $S_{\max}$. Therefore, in an optimal solution all cones are active and do not interfere in the range $[0, S_{\max}/2]$, but only every other cone in a chain can extend to $S_{\max}$. We say that two cones are *adjacent* if their L^∞-distance is $1/2$.

Variable gadgets. A variable gadget consists of a cyclic chain of an even number of adjacent cones. By the above observation this chain contributes maximally to H if every second cone is active in the range $[0, S_{\max}]$ and the remaining cones in the range $[0, S_{\max}/2]$. Numbering the cones clockwise starting with the leftmost cone in the top row of the gadget, we denote the state where the odd cones extend to the full scale $S_{\max}$ as *true* and the state where the even cones extend to $S_{\max}$ as *false*; see Figure 6.5.

Literal gadgets. Each variable gadget has indentations. Their number depends on how often the variable occurs in the clauses of φ. At each indentation we can connect a leg of a three-legged comb that serves as a literal gadget. Each leg consists of an even chain of adjacent cones leading towards the clause gadget. The middle leg of a comb is a simple vertical chain, whereas the left and right leg start vertically and then bend 90 degrees in order to reach the clause gadget horizontally. For a positive literal the leg is adjacent to the beginning (in clockwise order) of the indentation, for a negative literal the leg is adjacent to the end of the indentation, see Figure 6.5. Thus, if a literal evaluates to *false*, the corresponding literal leg must start with a cone of height $S_{\max}/2$; otherwise it can start with a cone of height $S_{\max}$. A literal leg contributes maximally to H if every other cone reaches $S_{\max}$. Hence it has two maximal states—either the odd or the even cones reach $S_{\max}$.

Clause gadgets. The clause gadget in Figure 6.6 consists of three pairwise adjacent cones, that is, at most one of them can extend to $S_{\max}$, and their maximal total contribution to H is $2S_{\max}$. Each of the clause cones is adjacent to the last cone of one of the three incoming literal legs. Since the literal legs consist of an even number of cones, the leg for a *true* literal ends in a cone of height $S_{\max}/2$, while a *false* literal leg ends in a cone of height $S_{\max}$—at least if the literal legs contribute maximally to H. Thus, none of the three clause cones can be of height $S_{\max}$ if and only if all literals in the clause are *false*.

We claimed above that literal gadgets always consist of an even number of cones. This ensures that in both states, they contribute the same amount to H. Now assume that

a specific literal gadget actually consists of an odd number of cones. Then we replace a straight part consisting of twelve cones by the parity inverter gadget depicted in the top part of Figure 6.7. For comparison, the bottom part of Figure 6.7 shows a straight chain of cones. Note that the inverter gadget and the straight chain span the same distance on the grid; the inverter with an odd number of cones, the chain with an even number.

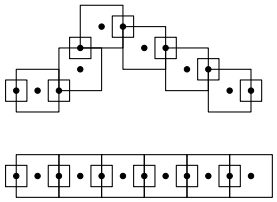


Figure 6.7: Parity inverter gadget (top).

Reduction. As before we determine the threshold K such that φ is satisfiable if and only if $H(\mathcal{S}(\mathcal{E}_\varphi)) \geq K$ and all variable, literal, and clause gadgets contribute maximally to H. Variable and literal gadgets contribute maximally if they correctly encode their truth values and clause gadgets contribute maximally if at least one of their literals evaluates to true and thus the clause is satisfied.

We denote the number of cones for the variable gadget of variable x by n_x and the number of cones for the literal leg of literal λ by m_λ. These numbers depend on φ and are fixed in the construction. Let

$$K = \frac{3}{4} \cdot S_{\max} \sum_{v \in \mathrm{Var}(\varphi)} n_v + \frac{3}{4} \cdot S_{\max} \sum_{\lambda \in \mathrm{Lit}(\varphi)} m_\lambda + 2m \cdot S_{\max},$$

where $\mathrm{Var}(\varphi)$ and $\mathrm{Lit}(\varphi)$ denote the variables and literals in φ, respectively. The summands of K correspond to the maximum contributions of all variable gadgets, literal gadgets, and clause gadgets. By the above observation on maximal contributions a total active range height $H(\mathcal{S}(\mathcal{E}_\varphi))$ of at least K can be achieved by an optimal solution $\mathcal{S}(\mathcal{E}_\varphi)$ only if all clause gadgets contribute maximally and thus if φ is satisfiable.

Since the variable-clause graph of φ can be drawn on a grid of size $O(m^2)$ [KR92], the set $\mathcal{E}_\varphi$ also consists of $O(m^2)$ extrusions placed on a grid of size $O(m^2)$ and the reduction takes polynomial time. □

6.3 Algorithmic Toolbox

We give a toolbox of six different algorithms to tackle several variants of 1d- and 2d-ARO problems. Some are only briefly covered since they are based on well-known techniques: dynamic programming, a left-to-right greedy algorithm, line stabbing, and divide and conquer. We concentrate on two algorithms that apply new techniques: a top-to-bottom fill-down sweep, and a level-based small-to-large greedy algorithm.

6.3.1 Dynamic Programming

Triangles

We start by considering simple 1d-ARO with proportional dilation: each extrusion E_i is a triangle with apex on the x-axis and top edge on the horizontal line $s = S_{\max}$. The truncated extrusions T_i differ only by having (possibly) lower top edges. Observe that in an optimal solution at least one T_i has height $S_{\max}$, and thus divides the problem into two independent subproblems. This is the essence of the dynamic programming solution.

Theorem 6.4 *Simple 1d-ARO with proportional dilation can be solved in $O(n^3)$ time using $O(n^2)$ space.*

Proof. Let p_i be the apex of the triangular extrusion E_i on the x-axis. For ease of notation define dummy triangles E_0 and E_{n+1} with apexes p_0 and p_{n+1}, and assume that $p_0, \ldots, p_{n+1}$ are sorted from left to right. For $i < j$, define the *free space* $\Delta(i, j)$ between p_i and p_j to be the triangular or trapezoidal space enclosed by $s = 0$, the right side edge of E_i, the left side edge of E_j, and possibly $s = S_{\max}$. Let $\mathcal{A}[i, j]$ $(i < j)$ be the optimal solution for $p_{i+1}, \ldots, p_{j-1}$ in $\Delta(i, j)$. In $\mathcal{A}[i, j]$, at least one of $T_{i+1}, \ldots, T_{j-1}$ must touch a non-bottom edge of $\Delta(i, j)$, thus dividing the problem into two independent subproblems. For each $k = i + 1, \ldots, j - 1$, we denote by $s_k^{i,j}$ the scale at which T_k first reaches a non-bottom edge of $\Delta(i, j)$. We initialize $\mathcal{A}[i, i + 1] = 0$ for $i = 0, \ldots, n$ and recursively compute

$$\mathcal{A}[i, j] = \max\{\mathcal{A}[i, k] + s_k^{i,j} + \mathcal{A}[k, j] \mid i + 1 \leq k \leq j - 1\}.$$

Obviously, the optimal solution for our problem is $\mathcal{A}[0, n + 1]$. Each of the $O(n^2)$ entries in the dynamic programming table is computed in $O(n)$ time, resulting in a total running time of $O(n^3)$. □

6.3.2 Left-to-Right Greedy algorithm

Unit-Height Rectangles

Van Kreveld et al. [vKSW99] presented the following greedy algorithm for maximum independent set (MIS) among axis-aligned rectangles of unit height. We are given a set $\mathcal{E}$ of unit-height rectangles. Until $\mathcal{E}$ is empty, repeatedly select the rectangle $E \in \mathcal{E}$ with leftmost right edge, and remove from $\mathcal{E}$ all remaining rectangles intersecting E. This takes $O(n \log n)$ time, and is a $(1/2)$-approximation for MIS [vKSW99].

For ARO the same algorithm yields a $(1/3)$-approximation, for the following reason. If an active range of some extrusion E_{opt} in an optimal solution is not active in the solution $\mathcal{A}$ of the algorithm, there must be an extrusion $E_{\text{alg}} \neq E_{\text{opt}}$ that is selected in $\mathcal{A}$ such that the right edge of E_{alg} intersects E_{opt}; therefore E_{opt} has been removed from $\mathcal{E}$ in the algorithm and is inactive in $\mathcal{A}$. But since all extrusions are unit-height rectangles and E_{alg} has the leftmost right edge when it is selected, E_{alg} can only be responsible for removing rectangles with a total active range height of less than three times the height of E_{alg}.

Theorem 6.5 *The maximum total active range height for a set of n rectangular extrusions of unit height can be approximated within a factor of $1/3$ in $O(n \log n)$ time.*

6.3.3 Line Stabbing

We use line stabbing for unit squares and unit-width rectangles, that is, general 1d-ARO with constant dilation and equal-size labels. Line stabbing is a special case of the shifting

technique by Hochbaum and Maass [HM85]. The idea is to stab all extrusions with vertical lines of distance at least 1 such that each extrusion is stabbed by exactly one line and each line stabs at least one extrusion. The stabbing lines are numbered l_1 to l_k from left to right. Since all extrusions have unit width, those intersecting l_i do not overlap those intersecting l_{i+2}. Our approximate solutions make use of the *optimal* solutions for either the extrusions stabbed by a single line or those stabbed by two adjacent lines.

First, we show how to compute the optimal solution on a single stabbing line. The exact algorithm for a single stabbing line applies more generally to a set of arbitrarily sized rectangles. This will be used again in Section 6.3.4.

Lemma 6.1 *The maximum total active range height of a set $\mathcal{E}$ of n rectangles stabbed by a single vertical line l can be computed in $O(n \log n)$ time.*

Proof. The algorithm basically solves a one-dimensional problem on l. Sweep a horizontal line from bottom to top over $\mathcal{E}$, stopping at the event points s_i and S_i for each $E_i \in \mathcal{E}$. Maintain a stack, initially empty, and a current rectangle, initially null. At the event point s_i, push E_i onto the stack, and if the current rectangle is null then make E_i the current rectangle and set its active range $[a_i, A_i] = [s_i, S_i]$. At the event point S_i, if E_i is the current rectangle then pop rectangles from the stack until a rectangle E_j is popped with $S_j > S_i$. If such an E_j is found then make E_j the current rectangle and set its active range $[a_j, A_j] = [S_i, S_j]$; otherwise reset the current rectangle to null.

The way the algorithm works, the current rectangle is never null whenever the sweep line intersects some rectangle in $\mathcal{E}$. So obviously the total active range height of the algorithm's solution is optimal since it equals the height of the union of $\mathcal{E}$. Since every rectangle is pushed onto and popped from the stack once, the sweep takes linear time, but we need $O(n \log n)$ time to sort the event points initially. □

Unit-Width Rectangles

For unit-width rectangles we partition the vertical stabbing lines into sets Λ_1 and Λ_2, containing all the stabbing lines with odd and even indices, respectively. By Lemma 6.1 the solution for each individual stabbing line, and thus also the solution $\mathcal{A}_i$ for all rectangles intersecting lines in Λ_i, can be computed optimally. From the candidate solutions $\mathcal{A}_1$ and $\mathcal{A}_2$ we choose the one maximizing H as our approximate solution $\mathcal{A}$. Each solution $\mathcal{A}_i$ is at least as good as the optimal solution restricted to the rectangles stabbed by Λ_i. Hence $\mathcal{A}$, the better of the two solutions $\mathcal{A}_1$ and $\mathcal{A}_2$, is at least half as good as the optimal solution. The approximate solution $\mathcal{A}$ can be computed in $O(n \log n)$ time using Lemma 6.1. Thus we obtain the following.

Theorem 6.6 *The maximum total active range height for a set of n rectangular extrusions of unit width can be approximated within a factor of $1/2$ in $O(n \log n)$ time.*

Unit Squares

To obtain a better approximation for the case that all extrusions are unit squares, we partition the vertical stabbing lines into *three* sets, $\Lambda_i = \{l_j \mid j = i \pmod 3\}$ for $i = 1, 2, 3$. Deleting all squares stabbed by one of the sets Λ_i divides the problem into independent subproblems defined by two consecutive stabbing lines each. A greedy sweep-line algorithm finds the optimal solution for each of these subproblems as follows.

Lemma 6.2 *The maximum total active range height of a set $\mathcal{E}$ of n unit squares stabbed by two vertical lines of distance at least 1 can be computed in $O(n \log n)$ time.*

Proof. The algorithm sweeps a line from bottom to top over $\mathcal{E}$, stopping at event points s_i and S_i for each $E_i \in \mathcal{E}$. We maintain two priority queues B^{left} and B^{right} of available squares on each stabbing line, where B^{left} allows to extract the leftmost and B^{right} the rightmost square. Furthermore, we maintain an active set $\mathcal{L}$ of at most two independent squares, one on the left and one on the right stabbing line.

In the beginning all sets are empty. At the event point s_i, insert E_i into list B^{left} or B^{right}, depending on which line stabs E_i. If E_i is the leftmost square in B^{left} and it does not intersect the currently active square on the right stabbing line then replace the currently active square E_j on the left stabbing line by E_i. This means we set $A_j = s_i$ and $a_i = s_i$. If there is no currently active square on the right stabbing line and the rightmost square E_k in B^{right} and E_i are independent then add E_k to $\mathcal{L}$ and set $a_k = s_i$. The case that E_i is the rightmost square in B^{right} is handled analogously.

At the event point S_i we remove E_i from its list B^{left} or B^{right}. If E_i is one of the active squares in $\mathcal{L}$ then it is also removed from $\mathcal{L}$ and we set $A_i = S_i$. Furthermore, depending on the stabbing line of E_i, we insert the leftmost (or rightmost) square E_j from B^{left} (or B^{right}) as the successor of E_i into $\mathcal{L}$ if it does not intersect the other currently active square in $\mathcal{L}$. In that case we set $a_j = S_i$.

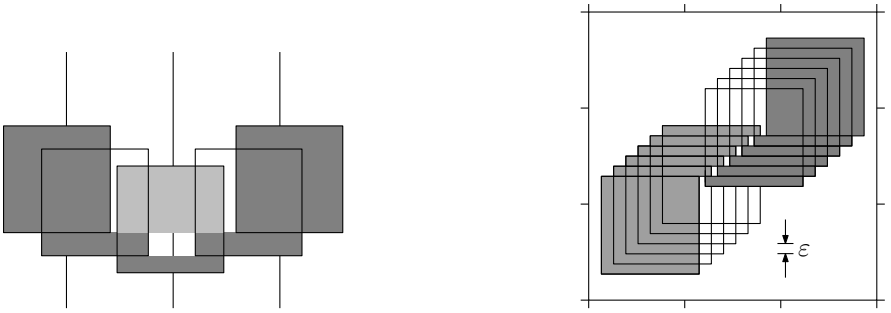
At each point in time $\mathcal{L}$ contains the leftmost square on the left stabbing line and the rightmost square on the right stabbing line—if they are independent. Thus the algorithm activates the maximum number of independent squares (at most two) at each scale. It remains to show that the active range is contiguous for each square. Suppose a square E_i leaves $\mathcal{L}$ before the sweep line reaches S_i. Then it must have been replaced by a more extremal (left or right) square E_j. Since all square have unit height we know that $S_j > S_i$ and thus E_i will never again be extremal and active. This shows that the solution of the algorithm is valid.

Sorting the event points and maintaining the priority queues using a heap data structure takes $O(n \log n)$ time. □

Using the algorithm described above, we optimally solve the three subproblems created by removing the squares stabbed by, respectively, one of the sets Λ_i. The solution to each of the three subproblems is at least as good as the restriction of the globally optimal solution to the squares stabbed in the respective subproblem. Thus by the pigeon-hole principle, the best of the three subsolutions is a (2/3)-approximation. This yields the following theorem.

Theorem 6.7 *The maximum total active range height for a set of n unit-square extrusions can be approximated within a factor of $2/3$ in $O(n \log n)$ time.*

Note that the above technique can *not* be easily extended to a polynomial-time approximation scheme by defining sets $\Lambda_1, \ldots, \Lambda_k$ for some $k > 3$ since it is not clear how to solve the remaining $(k-1)$-line subproblems (near-) optimally in polynomial time. Unlike the situation in Lemma 6.2, greedily activating the maximum number of independent squares at each scale for $k > 3$ can result in a square being activated more than once, that is, its active range is not a contiguous interval. This is illustrated in Figure 6.8a, where the square in the middle has an invalid non-contiguous active range. Partitioning the plane into square blocks (as Hochbaum and Maass [HM85] do) instead of vertical strips does not help either since any optimal solution inside a $(t \times t)$-block may contain active ranges from an arbitrary number of unit squares; see Figure 6.8b, where in the depicted optimal solution each of the n squares has a non-empty active range. Any k-element subset $\mathcal{E}'$ of the depicted n-square instance $\mathcal{E}$ has $H(\mathcal{E}') \approx 3 + k/n$, whereas $H(\mathcal{E}) = 4 - \Theta(\varepsilon)$.



(a) Subinstance defined by three consecutive stabbing lines.

(b) Optimal solution of an instance with n squares in a (3×3)-block.

Figure 6.8: Two examples that prevent us from giving a PTAS for unit squares. Active ranges are shaded.

6.3.4 Divide and Conquer

Arbitrary Rectangles

Agarwal et al. [AvKS98] gave an $O(n \log n)$-time divide-and-conquer algorithm to compute a $(1/\log n)$-approximation for MIS among axis-aligned rectangles. Their algorithm readily adapts to ARO. First, the given set $\mathcal{E}$ of rectangles is split at the vertical line $\ell : x = x_{\text{med}}$, where x_{med} is the median of the at most $2n$ abscissae of the left and right edges of the given rectangles. This yields the three disjoint subsets $\mathcal{E}_{\text{left}}$, $\mathcal{E}_{\text{right}}$, and $\mathcal{E}_\ell$ of the rectangles in $\mathcal{E}$ lying left of ℓ, lying right of ℓ, and intersecting ℓ, respectively. The solutions $\mathcal{A}_{\text{left}}$ and $\mathcal{A}_{\text{right}}$ for $\mathcal{E}_{\text{left}}$ and $\mathcal{E}_{\text{right}}$ are computed recursively. The solution $\mathcal{A}_\ell$ for $\mathcal{E}_\ell$ can be computed optimally by the simple greedy algorithm given in Lemma 6.1. Since $\mathcal{A}_{\text{left}}$ and $\mathcal{A}_{\text{right}}$ are independent, the algorithm returns among the two solutions $\mathcal{A}_{\text{left}} \cup \mathcal{A}_{\text{right}}$ and $\mathcal{A}_\ell$, the one with the larger value of the objective function. The inductive proof of the following theorem is identical to that in [AvKS98] with the only difference that we compute an optimal set of active ranges for the one-dimensional instance $\mathcal{E}_\ell$ (using Lemma 6.1) instead of a maximum independent set.

Theorem 6.8 *The divide-and-conquer algorithm computes in $O(n \log n)$ time a $(1/\log n)$-approximation to the maximum total active range height for a set of n rectangles.*

6.3.5 Top-to-Bottom Fill-Down Sweep

A number of variants of 1d- and 2d-ARO are approximated within a constant factor by Algorithm 6.1, which follows below. The idea is to sweep a line or plane downwards over the extrusions in $\mathcal{E}$, and if $E_i \in \mathcal{E}$ is selected at scale s, we "fill" E_i from s down to its bottom—that is, we set $[a_i, A_i] = [s_i, s]$. Thus we have $a_i = s_i$ for every E_i that contributes to the objective function H. This effect may even be desired in the application since labels never disappear when zooming in—unless their available range ends.

We say that E_i is *available* if its available range contains the current sweep scale s, and *active* if its active range has already been set and contains s. Let the *trace* of an extrusion be its intersection with the sweep line (or plane). We consider the intersection graph of the traces, to which we refer as *conflict graph*. We are interested in scales where the conflict graph changes. We refer to such scales as *event points*. The set of event points contains all values of types S_i and s_i; some extrusion shapes have further event points. Due to the

types of dilation we consider, two extrusions E_i and E_j may define at most one further event point $s_{ij} \notin \{s_i, s_j, S_i, S_j\}$; this is the scale where the side edges or faces of E_i and E_j intersect. Let k be the number of type-s_{ij} events induced by $\mathcal{E}$.

We make use of a subroutine, `TryToPick(`E_i`,`s`)`, which means "if E_i does not intersect the interior of any extrusion already chosen to be active at the current sweep scale s, then make E_i active and set $[a_i, A_i] = [s_i, s]$".

Algorithm 6.1: Top-to-bottom fill-down sweep

Input: set $\mathcal{E} = \{E_1, \ldots, E_n\}$ of extrusions with available range $[s_i, S_i]$ for each E_i
Output: set $\mathcal{T} = \{T_1, \ldots, T_n\}$ of non-overlapping truncated extrusions $T_i \subseteq E_i$ with active range $[a_i, A_i]$ for each T_i

```
sweep a line or plane that is perpendicular to the s-axis from top to bottom
foreach event point s of type s_i, S_i, or s_ij do
    foreach available but inactive extrusion E_j in non-increasing order of S_j do
        TryToPick(E_j,s)
```

Next, we describe a generic implementation of Algorithm 6.1 that uses three data structures: an *event queue*, an *active set*, and an *available set*. The event queue maintains the events in the right order and supports queries for the next event. The events are sorted by non-increasing scale and such that—whenever multiple events occur at the same scale—the events of type s_i come first, then the events of type s_{ij}, and finally the events of type S_i. The active set is a data structure that maintains all extrusions that are active at the current scale of the sweep line (or plane). Given an inactive extrusion to be activated, the active set either adds it as a new element or reports a conflict with another extrusion that is already active. The available set stores all extrusions that are available at the current scale. It supports returning the available extrusions E_i sorted non-increasingly by their S_i-values.

Lemma 6.3 *A generic implementation of Algorithm 6.1 runs in $O(n^2(k+n))$ time and takes $O(k+n)$ space.*

Proof. We use a heap for the event queue that is initialized with the $2n$ events of type s_i and S_i. The k events of type s_{ij} are added during the sweep. It thus takes $O((k+n)\log n)$ time and $O(k+n)$ space to add all $O(k+n)$ events during the algorithm. The active set is an unordered list of extrusions; each query with a new candidate extrusion E simply traverses the $O(n)$ active extrusions and tests each of them for intersection with E in constant time. If E does not intersect any active extrusion, it is appended to the list. Deleting an element from the active set takes $O(n)$ time. The available set is implemented as a binary search tree storing the extrusions E_j ordered non-increasingly by their upper available scale S_j. Each insertion and deletion takes $O(\log n)$ time.

We iterate through the events as provided by the event queue. At an event of type s_i we remove E_i from either active set or available set in $O(n)$ time. At an event of type S_i we first insert E_i into the available set in $O(\log n)$ time. Then we traverse the $O(n)$ extrusions in the active set and in the available set and check each of them for a side-edge (or side-face) intersection with E_i. For each intersection found, we add the corresponding event of type s_{ij} into the event queue. Next, at each event, regardless of its type, we traverse the $O(n)$ extrusions in the available set and try to pick each of them in order. Each call to the try-to-pick routine has to check $O(n)$ extrusions in the active set for a

possible intersection. Hence this implementation runs in $O(n^2(k+n))$ time and $O(k+n)$ space independent of the extrusion type. □

Let $\mathcal{A} = \{(a_i, A_i) \mid E_i \in \mathcal{E}\}$ be the solution computed by Algorithm 6.1. We say that an extrusion E_j *blocks* another extrusion E_i at scale s under a given solution if E_i and E_j overlap (that is, their interiors intersect) at s and $s \in [a_j, A_j]$. Note that this implies that $s \notin [a_i, A_i]$. We say that two extrusions are *independent at* s if their traces at scale s are non-overlapping. The following lemma will help prove all approximation factors in this section.

Lemma 6.4 *If, for any set $\mathcal{E}$ of extrusions, for any $E \in \mathcal{E}$, and for any scale $s \geq 0$, E blocks no more than c pairwise independent extrusions at s, then Algorithm 6.1 computes a $(1/c)$-approximation for the maximum total active range height.*

Proof. Suppose that $E \in \mathcal{E}$ is inactive at scale s under $\mathcal{A}$. Then E must be blocked at the nearest event point above (or at) s since otherwise it would be picked by Algorithm 6.1. Since the conflict graph changes only at event points, E is blocked at s. Thus, in $\mathcal{A}$, if E is inactive at any scale s then E is blocked at s.

If at any scale no extrusion can block more than c mutually independent extrusions, and in $\mathcal{A}$ every inactive extrusion is blocked, then at any scale the number of active extrusions in an optimal solution can be no more than c times the number in $\mathcal{A}$. Integrating over all scales proves the lemma. □

For each of the extrusion shapes covered in this section we determine a value for c, usually 2 or 4. For example, it is easy to see that $c = 2$ holds for unit-width rectangles (or, more generally, for any set of rectangles where the x-order of the left edges is the same as the x-order of the right edges), so Algorithm 6.1 yields a (1/2)-approximation. (Theorem 6.6 states the same result via a different algorithm based on line stabbing.)

We can improve the running time of Algorithm 6.1 over the generic implementation of Lemma 6.3 by using enhanced, extrusion-specific data structures for active set and available set that support the query operations described in the following implementation. The crucial idea for speeding up the implementation is to quickly find an available extrusion that can be activated rather than testing at every event all available extrusions.

We iterate through the events provided by the event queue. Depending on the type of the current event we apply one of the following three procedures.

- For an event of type s_i we remove extrusion E_i either from the active set or from the available set. If E_i was an active extrusion we determine the region of the current sweep line or plane that had been blocked by E_i and query the available set for an available extrusion E_j within that region that has largest S_j value. Extrusion E_j—if it exists—is removed from the available set and added to the active set. Its active range $[a_j, A_j]$ is set to $[s_j, s_i]$. Depending on the parameter c, which determines how many independent extrusions can be blocked by a single extrusion (see Lemma 6.4), we need to repeat the query for another available extrusion up to c times with appropriately modified query regions.

- For an event of type S_i we query the active set for conflicts with the new extrusion E_i. If it does not intersect any active extrusion, we add E_i to the active set and assign the active range $[a_i, A_i] = [s_i, S_i]$. Otherwise we add E_i to the available set.

- For an event of type s_{ij} such that one of the two extrusions E_i and E_j, say E_i, is active, we try to add the other by querying the active set for conflicts. If there are no conflicts, we remove E_j from the available set and add it to the active set. In this case its active range is set to $[a_j, A_j] = [s_j, s_{ij}]$.

We need to be careful if multiple events of type s_i share the same scale s. It is important that we remove all extrusions that become inactive at s from the active set *before* we query the available set for new extrusions to activate. We build a list of the regions freed on the sweep line (or plane) by each removed extrusion at scale s such that overlapping regions are merged. Subsequently, we query the available set with each of those regions. Similarly, for multiple events of type s_{ij} at the same scale s (such that E_i is active and E_j is inactive), we first build a list of the inactive extrusions E_j sorted non-increasingly by the S_j-values. Afterwards, we try to pick these extrusions in the given order.

The implementation described above yields the following lemma.

Lemma 6.5 *Algorithm 6.1 can be implemented performing $O(n)$ insertions into available set and active set, $O(n)$ deletions from available set and active set, $O(cn)$ range queries to the available set, and $O(k+n)$ conflict queries to the active set.*

The actual running time still depends on the insertion, deletion, and query times of the extrusion-specific data structures used for active set and available set. In the remainder of this subsection we discuss different extrusion types. For each type we analyze the performance of Algorithm 6.1 in terms of approximation factor as well as time and space complexity.

Congruent Trapezoids

In this part we consider the 1d-ARO problem for congruent trapezoids as the extrusion shapes.

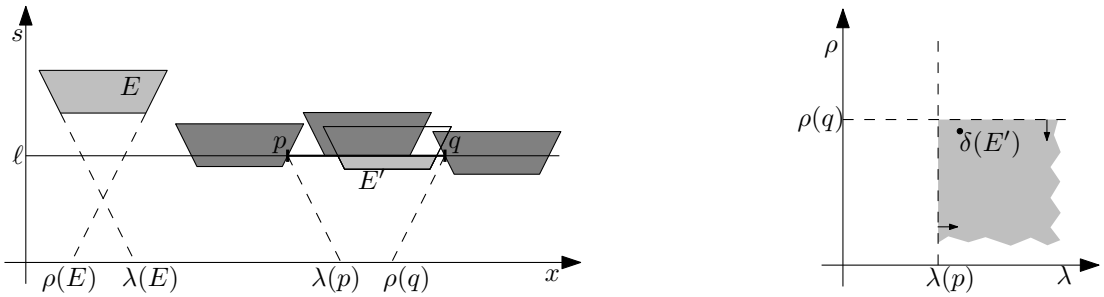
Lemma 6.6 *Algorithm 6.1 approximates the maximum total active range height of a set of n congruent trapezoids within a factor of $1/2$.*

Proof. We show that if a trapezoid E_j blocks another trapezoid E_i at scale s under solution $\mathcal{A}$, then E_i must intersect at least one side edge of E_j at scale s. This implies $c = 2$ in Lemma 6.4, and thus yields the approximation factor of $1/2$.

Suppose E_j blocks E_i at s. If $S_i \leq S_j$ then E_i is at least as wide as E_j at s and it must intersect a side edge of E_j. Otherwise, if $S_i > S_j$ then E_i is available at scale A_j, too, when Algorithm 6.1 selects E_j, and since the trapezoids are wider at higher scales, E_i and E_j also intersect at scale A_j. Recall that the algorithm selects the extrusions in non-increasing order of the scale of their top edges. Since it picks E_j and not E_i, E_i must be blocked by another trapezoid at scale A_j that does not block E_j. Thus, E_i must intersect a side edge of E_j at scale A_j, and since they are congruent, it still intersects a side edge at scale s. □

In order to improve the time and space complexity of Algorithm 6.1 for congruent trapezoids over Lemma 6.3 we first describe an implementation of the event queue that is more space efficient.

Lemma 6.7 *For 1d-ARO problems the event queue can be implemented to iterate through all events in $O((k+n)\log n)$ time and $O(n)$ space.*



(a) The (λ, ρ)-coordinates of a trapezoid E. Trapezoid E' can be activated in the free interval $[p, q]$ on ℓ.

(b) Query region for finding $\delta(E')$ in (λ, ρ)-space.

Figure 6.9: Transformation to (λ, ρ)-coordinates.

Proof. As before the event queue is implemented as a heap. It is initialized with the $2n$ events of types s_i and S_i for all E_i in $\mathcal{E}$. This takes $O(n \log n)$ time. During the sweep we simultaneously detect events of type s_{ij} that lie ahead and insert them into the event heap using a standard algorithm for finding line segment intersections [dBCvKO08, Theorem 2.4]. This algorithm stores the horizontal order of the line segments (in our case the side edges of the trapezoids) intersecting the sweep line and considers only intersections between neighboring segments in this order. Therefore, it is possible that type-s_{ij} events are inserted into and removed from the event heap several times whenever the horizontal order of the segments changes. Still, all insertions and deletions in the event heap take $O((k + n) \log n)$ time. By detecting the events of type s_{ij} on the fly at most $O(n)$ such events are stored in the heap in each step of the algorithm. □

We give the running time and space complexity of the algorithm in the following lemma by using the event queue of Lemma 6.7 and describing the data structures used for active set and available set.

Lemma 6.8 *Given a set of n congruent trapezoids, Algorithm 6.1 can be implemented to run in $O(k \log n + n \log^2 n)$ time and $O(n \log n)$ space, where k is the number of side-edge intersections between pairs of trapezoids.*

Proof. Our implementation uses the event queue of Lemma 6.7 to iterate through the events. The active set is implemented as a balanced binary search tree in which the active trapezoids are ordered from left to right by their (disjoint) intersections with the sweep line. Thus insertions, deletions, and conflict queries all take $O(\log n)$ time.

For an efficient implementation of the available set we take advantage of the fact that all trapezoids are congruent. This allows us to transform them into a dual space. We assume that all trapezoids lie above the x-axis. For a trapezoid E we denote the abscissae of the intersections of the x-axis and the downward extensions of the left and right edges of E by $\lambda(E)$ and $\rho(E)$, respectively. This defines a point $\delta(E) = (\lambda(E), \rho(E))$ in the dual space. Similarly, we map an arbitrary point p above the x-axis to a point $\delta(p)$ in the dual space using the abscissae of the intersections of the x-axis and lines with the same slopes as the left and right edges of the trapezoids. Figure 6.9 shows the transformation into the dual space. Most importantly, note that in Figure 6.9a we can activate the trapezoid E' in the free space defined by the line segment $[p, q]$ if $\lambda(E') \geq \lambda(p)$ and $\rho(E') \leq \rho(q)$.

Now we can implement the available set as a two-dimensional *dynamic priority range tree*. A dynamic priority range tree is a dynamic range tree [dBCvKO08, Chapter 5.3], which

stores the points $\delta(E)$ for all available but inactive trapezoids E. It consists of a first-level tree based on the λ-coordinates and for each of the nodes of that tree a second-level tree based on the ρ-coordinates. The difference from a standard range tree is that each node in a second-level tree is augmented with a pointer to the leaf in its subtree that has highest priority, in our case has largest S_i-value. A range query on this data structure asks for the point $\delta(E_i)$ with maximum S_i among all points within a rectangular region in (λ, ρ) space. Such a query for the single point with highest priority in the query region takes $O(\log^2 n)$ time—in contrast to $O(\kappa + \log^2 n)$ time to report all κ points in that region. In particular, a query with the region $[\lambda(p), \infty) \times (-\infty, \rho(q)]$ returns the trapezoid E_i with largest S_i-value that can be activated in the free space defined by the line segment $[p, q]$, see Figure 6.9b. Inserting E_i into the active set splits $[p, q]$ into two parts $[p, p']$ and $[q', q]$ to the left and right of E_i. We also query the available set with the regions corresponding to these two intervals in order to find a second independent extrusion to activate. Since an extrusion can block at most two other independent extrusions we add at most two extrusions to the active set and do not need more queries. Insertions and deletions in the available set take $O(\log^2 n)$ time each.

Since each operation on the active set takes $O(\log n)$ time and each operation on the available set takes $O(\log^2 n)$ time, the total running times add up to $O((k + n) \log n)$ time for the active set and $O(n \log^2 n)$ time for the available set according to Lemma 6.5. The active set uses $O(n)$ space, and the available set uses $O(n \log n)$ space. Together with the event queue of Lemma 6.7 this yields the desired time and space bounds. □

Combining Lemmas 6.6 and 6.8 yields the following.

Theorem 6.9 *A $(1/2)$-approximation for the maximum total active range height of a set of n congruent trapezoids can be computed in $O(k \log n + n \log^2 n)$ time and $O(n \log n)$ space, where k is the number of side-edge intersections between pairs of trapezoids.*

Congruent Frusta

Axis-aligned congruent square frusta are the 2d-ARO analogues of the congruent trapezoids in the previous paragraph. Here, a blocked frustum must intersect a side *face* of its blocker. The number of independent frusta that can intersect a single face depends on W, the ratio of the length of the top edges of each frustum to the length of the bottom edges.

Theorem 6.10 *Algorithm 6.1 computes a $1/(4W)$-approximation for the maximum total active range height of a set of n axis-aligned congruent square frusta in $O(n^2(k + n))$ time and $O(k+n)$ space, where k is the number of side-face intersections between pairs of frusta.*

Proof. We first show that if a frustum E_j blocks another frustum E_i at scale s under solution $\mathcal{A}$, then E_i must intersect at least one side face of E_j at scale s. So suppose E_j blocks E_i at s. If $S_i \leq S_j$ then E_i is at least as large as E_j when considering their intersections with the sweep plane at scale s, so E_i must intersect a side face of E_j. Otherwise, if $S_i > S_j$ then E_i is available at scale A_j, too, when Algorithm 6.1 selects E_j, and since the frusta are larger at higher scales, E_i and E_j also intersect at A_j. Since the algorithm picks E_j and not E_i (which it considers first), E_i must be blocked by another frustum at A_j that does not block E_j. Thus, E_i must intersect a side face of E_j at A_j, and since the extrusions are congruent, it still intersects a side face of E_j at scale s.

This implies that an extrusion E can block at most $4W$ independent extrusions at any scale s.Thus the approximation factor of $1/(4W)$ follows from Lemma 6.4. Running time and space requirements follow directly from Lemma 6.3. □

Trapezoidal Segments of Congruent Underlying Triangles

Here we consider the 1d-ARO problem with proportional dilation, where the extrusions are trapezoidal segments of congruent underlying triangles that have their apexes on the x-axis.

Theorem 6.11 *Algorithm 6.1 computes a* $(1/2)$*-approximation for the maximum total active range height of a set of* n *trapezoidal segments of congruent triangles in* $O((k+n)\log n)$ *time and* $O(n)$ *space, where* k *is the number of side-edge intersections between pairs of trapezoids.*

Proof. Since the underlying triangles are congruent and horizontally aligned, the width of every trapezoid is the same at each scale. This implies that any trapezoid blocked by another trapezoid E intersects a side edge of E. Thus, at most two extrusions blocked by E at scale s can be independent at s, and the approximation factor $1/2$ follows from Lemma 6.4.

In the implementation of the algorithm we use the event queue of Lemma 6.7. For the active set and available set data structures note that we have a natural representation of the extrusions as the x-values of the apexes of the underlying triangles. With this representation we can easily determine the query interval on the x-axis that contains all cones within a given interval $[p, q]$ at the current sweep scale. For a point p on the current sweep line let p_{left} and p_{right} be the apex locations on the x-axis of two hypothetical triangles having p on their left and right side edge, respectively. Figure 6.10 shows how the interval $[p, q]$ on the current sweep line corresponds to the interval $[p_{\text{left}}, q_{\text{right}}]$ on the x-axis.

We implement the active set as a balanced binary search tree storing the x-position of the active apexes. We implement the available set as a priority search tree storing the x-position of the apexes with the S_i-values as priority.

Insertions and deletions in the active set take $O(\log n)$ time each. A conflict query with a candidate extrusion E finds the two active extrusions to the left and to the right of E in $O(\log n)$ time and tests them for intersection with E at the current sweep scale in constant time.

Insertions and deletions in the available set also take $O(\log n)$ time each. A query region for the available set is an interval on the x-axis and the extrusion with highest priority in this interval is found in $O(\log n)$ time. Note that if an extrusion E_i is removed from the active set at event s_i, this yields a free interval $[p, q]$ on the sweep line at scale s_i defined by the two active neighbors of E_i, see Figure 6.10. Thus the query interval for the available set is $[p_{\text{left}}, q_{\text{right}}]$. If an extrusion E_j is found, let q' and p' be the left and right intersection points of E_j with the sweep line, respectively. In a second step we also need to query the available set with the intervals $[p_{\text{left}}, q'_{\text{right}}]$ and $[p'_{\text{left}}, q_{\text{right}}]$ on either side of E_j searching for a second independent extrusion to activate. In the example of Figure 6.10, extrusion E_k will be found in the query $[p'_{\text{left}}, q_{\text{right}}]$. Since an extrusion can block at most two other independent extrusions, we add at most two extrusions to the active set in this step, and we do not need more queries.

Since each operation on both the active set and the available set takes $O(\log n)$ time, Lemma 6.5 yields a total running time of $O((k+n)\log n)$. By Lemma 6.7 this is also the time needed to iterate through all events. All three data structures use $O(n)$ space. □

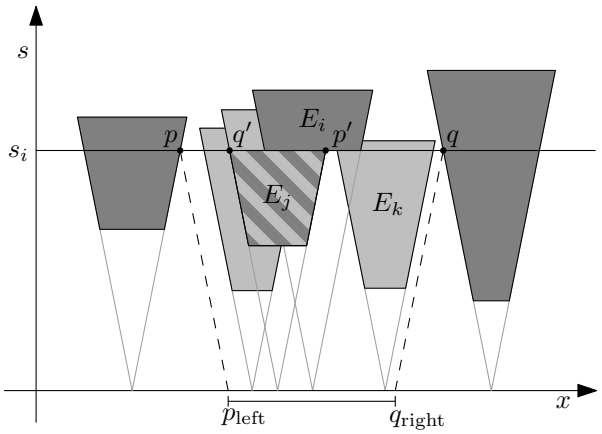


Figure 6.10: At the event s_i the available set is queried for the extrusion E_j with largest S_j-value in the interval $[p_{\text{left}}, q_{\text{right}}]$. Active extrusions are filled with dark gray, available extrusions with light gray.

Frustal Segments of Congruent Underlying Square Cones

In this part we are concerned with extrusions that are frustal segments of underlying axis-aligned congruent square cones with their apexes at $s = 0$. This is the 2d-ARO equivalent of trapezoidal segments of congruent triangles, which we have treated in Theorem 6.11. As in the previous section we can represent each extrusion by the position of the apex of its underlying cone.

Theorem 6.12 *Given a set of n frustal segments of axis-aligned congruent square cones, Algorithm 6.1 computes a $(1/4)$-approximation for the maximum total active range height in $O((k+n)\log^2 n)$ time and $O(k + n\log n)$ space, where k is the number of side-face intersections between pairs of frusta.*

Proof. Since the intersection of any extrusion with the sweep plane at scale s has the same constant size, we know that any extrusion blocked by an extrusion E at s must intersect one of the four corner edges of E at s. (Two side faces meet at a corner edge.) This means that at most four blocked extrusions can be independent and the approximation factor of $1/4$ follows from Lemma 6.4.

The implementation of Algorithm 6.1 uses an event queue realized as a heap that is initialized with the events of type s_i and S_i. The active set is implemented as a two-dimensional range tree, and the available set is implemented as a two-dimensional priority range tree (see Lemma 6.8). In both range trees the extrusions are represented by the apex positions of their underlying cones in the (x, y)-plane, where the priority for the available extrusions is their S_i-value.

Insertions and deletions in the range trees take $O(\log^2 n)$ time. Since the extrusions are square cones, the query region for a conflict query of the active set with an extrusion E at scale s is a square in the (x, y)-plane. This square contains the apex locations of all extrusions that intersect E at scale s and can be easily computed in constant time. Thus a conflict query takes $O(\log^2 n)$ time. Similarly, the query region for a range query to the available set at an event of type s_i is the square region in the (x, y)-plane that contains the apexes of all available extrusions that intersect E_i at scale s. If an extrusion E_j with highest priority is found, we perform a second query to the available set for the next extrusion that intersects E_i but not E_j at scale s. This query region is L-shaped and can

be composed of two rectangular range queries. Since each extrusion can block up to four independent extrusions, we need to perform up to four range queries to the available set at each event of type s_i, where each query is composed of a constant number of rectangular range queries. Thus each query to the available set also takes $O(\log^2 n)$ time.

Finally, at each event of type S_i, we query the active set and the available set for all extrusions that intersect a side face of E_i in the interval $[s_i, S_i]$. These extrusions are found using a query region that is a square annulus (the area between two concentric axis-aligned squares) and thus can be composed of four rectangular queries. Each intersection results in an event of type s_{ij} that is added to the event queue. In total, k such events are found in n queries that take $O(k + n \log^2 n)$ time. Since the event queue handles $O(k + n)$ events it has a running time of $O((k + n) \log n)$ and uses $O(k + n)$ space.

Thus Lemma 6.5 yields a running time of $O((k + n) \log^2 n)$. The three data structures use $O(k + n \log n)$ space. □

Simple ARO with axis-aligned congruent square cones is a special case of the above, where $[s_1, S_1] = \ldots = [s_n, S_n] = [0, S_{\max}]$, so we immediately get the following corollary.

Corollary 6.1 *Given a set of n axis-aligned congruent square cones, Algorithm 6.1 computes a $(1/4)$-approximation for the maximum total active range height in $O((k + n) \log^2 n)$ time and $O(k + n \log n)$ space, where k is the number of pairs of intersecting cones.*

6.3.6 Level-Based Small-to-Large Greedy Algorithm

In this section we give an algorithm for simple 2d-ARO with cones whose bases are axis-aligned squares. Rather than sweeping the events defined by the extrusions themselves, the algorithm in this section is based on intersecting the cones with $O(\log n)$ horizontal planes or *levels*. On each of these levels we activate some extrusions before we proceed to the next (lower) level. This yields a $(1/24)$-approximation for arbitrary cones and a $(1/4 - \varepsilon)$-approximation for congruent cones. The latter result is slightly worse than the $1/4$-approximation stated in Corollary 6.1, but the running time of the level-based algorithm is independent of k, the number of pairs of intersecting cones, which may be quadratic in n. We start with the general result.

Arbitrary Axis-Aligned Square Cones

Algorithm 6.2 below works in $O(\log n)$ phases, where phase i deals with the situation on a horizontal plane π_i at scale $\sigma_i = S_{\max}/2^i$. The phases are numbered 0 to $N_k = \lceil \log_\alpha kn \rceil$, where $\alpha = (k + 1)/k$ and $k \geq 1$ is an integer-valued parameter that will become important only when we treat congruent square cones below. Here we set $k = 1$, which yields $N_1 = \lceil \log_2 n \rceil$. For ease of presentation we add a dummy plane π_{N_k+1} at scale $s = 0$.

The trace of E_j at the scale σ_i of plane π_i is a square that we denote as E_j^i.We call E_j^i *active* if E_j is active at scale σ_i. Since we consider simple ARO, it holds that $a_1 = \cdots = a_n = 0$, and it remains to set $A_1, \ldots, A_n$. The algorithm works as follows.

When the algorithm terminates, all squares at level i that are inactive must intersect an active square—they are *blocked*. We associate each blocked square E_j^i to one of the active squares in the following way:

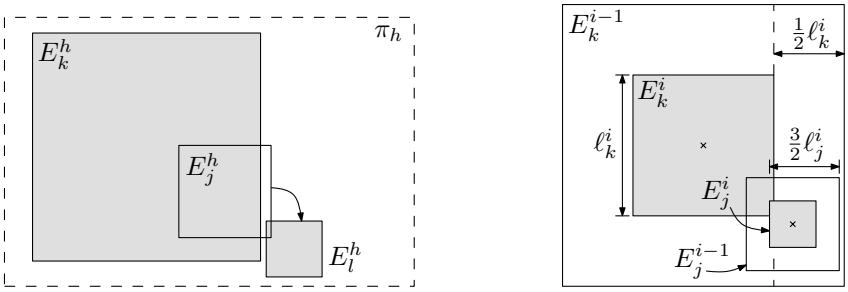
(i) if E_j^i was not blocked at the beginning of phase i but became blocked by a newly activated square E_k^i, then associate E_j^i to E_k^i;

Algorithm 6.2: Level-based algorithm for 2d-ARO

Input: set $\mathcal{E} = \{E_1, \ldots, E_n\}$ of axis-aligned square cones with available range $[s_i, S_i]$ for each E_i

Output: set $\mathcal{T} = \{T_1, \ldots, T_n\}$ of non-overlapping truncated extrusions $T_i \subseteq E_i$ with active range $[a_i, A_i]$ for each T_i

initialize all extrusions as inactive and set $[a_i, A_i] = [0, 0]$
for *phase* $i = 0$ **to** N_k **do**
  $\sigma_i \leftarrow S_{\max}/2^i$
  $C_i \leftarrow$ set of inactive squares in plane π_i that do not intersect any active square
  while $\mathcal{C}_i \neq \emptyset$ **do**
    $E_j^i \leftarrow$ smallest square in $\mathcal{C}_i$
    mark E_j^i as active and set $A_j = \sigma_i$
    remove E_j^i and all squares intersecting it from $\mathcal{C}_i$



(a) Square E_j^h is associated to E_l^h.

(b) E_k^{i-1} contains E_j^{i-1}.

Figure 6.11: Intersection behavior of E_j, E_k, and E_l at different levels.

(ii) if E_j^i was blocked in the beginning of phase i then associate E_j^i to any of its blocking squares that were active at the beginning of phase i.

Next, we show that the squares associated to an active square cannot be arbitrarily small.

Lemma 6.9 *Let E_k^i be an active square at level i with side length ℓ_k^i. Then any square associated to E_k^i has side length at least $\ell_k^i/3$ and intersects the boundary of E_k^i.*

Proof. Let E_j^i be a blocked square associated to E_k^i. If E_j^i is associated to E_k^i by case (i) above we know that the side length of E_j^i is larger than ℓ_k^i by the order in which Algorithm 6.2 selects the squares. There is nothing to show in this case.

So assume E_j^i is associated to E_k^i by case (ii) and let $h < i$ be the largest level in which E_j^h is *not* associated to E_k^h. (If there is no such level h then case (i) applies to E_j^0 and E_k^0 and the statement of the lemma holds.) Thus at level h we have E_j^h associated to some other active square E_l^h. Since E_k^{h+1} blocks E_j^{h+1} in the beginning of phase $h+1$ we know that E_k^h is already active and hence does not intersect E_l^h. On the other hand, both E_k^h and E_l^h must intersect E_j^h. This situation is depicted in Figure 6.11a.

Let ℓ_j^i be the side length of E_j^i and suppose $\ell_j^i < \ell_k^i/3$. Going to level $i-1$ the side lengths of the squares E_j^{i-1} and E_k^{i-1} are doubled, which means that E_j^{i-1} is fully contained

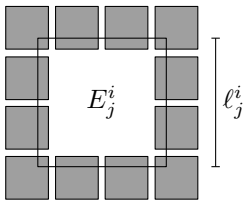


Figure 6.12: At most 12 pairwise independent squares of side length at least $\ell_j^i/3$ intersect E_j^i.

in E_k^{i-1}, see Figure 6.11b. This also holds for level $h \leq i-1$ and thus E_j^h cannot intersect the active square E_l^h, since E_l^h is disjoint from E_k^h—a contradiction. For the same reason E_j^i must intersect the boundary of E_k^i. □

We denote the active segments of the extrusions between planes π_{i-1} and π_i in the optimal solution $\mathcal{S}$ by $\mathcal{S}_i$ and in the solution $\mathcal{A}$ of our algorithm by $\mathcal{A}_i$, respectively. We charge the active range height $H(\mathcal{S}_i)$ to that of $H(\mathcal{A}_{i+1})$.

Lemma 6.10 *For each $i \in \{1, 2, \ldots, N_k - 1\}$ it holds that $H(\mathcal{A}_{i+1}) \geq H(\mathcal{S}_i)/24$.*

Proof. Let square E_j^i be active in $\mathcal{A}$ and consider the set $D(E_j^i)$ of squares in π_i associated to it. Clearly, the squares in $D(E_j^i)$ that correspond to active extrusions in $\mathcal{S}_i$ cannot intersect each other.

By Lemma 6.9, all squares in $D(E_j^i)$ have side length at least $\ell_j^i/3$ and intersect the boundary of E_j^i. Thus, at most 12 of those squares can be independent in π_i and hence active in $\mathcal{S}_i$ as in Figure 6.12. Now the height between levels i and $i-1$ is twice the height between levels $i+1$ and i. Hence the contribution of E_j to $H(\mathcal{A}_{i+1})$ is at least $1/24$ times the contribution to $H(\mathcal{S}_i)$ of the active extrusions in $\mathcal{S}_i$ whose squares at level i are associated to E_j^i. It follows that $H(\mathcal{A}_{i+1}) \geq 1/24\ H(\mathcal{S}_i)$.

Theorem 6.13 *Algorithm 6.2 computes a $(1/24)$-approximation for the maximum total active range height of a set of n axis-aligned arbitrary square cones in $O(n \log^3 n)$ time and $O(n \log n)$ space.*

Proof. In order to show the approximation factor it remains to compare $H(\mathcal{S}_{N_k}) + H(\mathcal{S}_{N_k+1})$ to $H(\mathcal{A}_{N_k+1}) + H(\mathcal{A}_1)$. The scale of π_{N_k-1} is at most $2S_{\max}/n$ and obviously there are at most n active cone segments in $\mathcal{S}$ below π_{N_k-1}, so their total active range height is at most $2S_{\max}$. On the other hand, there is at least one active cone segment in $\mathcal{A}_1$ of height $S_{\max}/2$ so that we can charge $H(\mathcal{S}_{N_k}) + H(\mathcal{S}_{N_k+1})$ to $H(\mathcal{A}_1)$. Together with Lemma 6.10 this implies the approximation factor of $1/24$.

For an efficient implementation of Algorithm 6.2 we store the squares in each level i in a two-dimensional segment tree τ_i, which supports deletion in $O(\log^2 n)$ time [dBCvKO08, Chapter 10.3]. For each square E_j^i that has been activated at a previous level we delete all intersecting squares as follows. Place vertices at each corner of E_j^i and two vertices on each edge equidistantly. Query τ_i with each of these 12 vertices and delete the returned squares from τ_i. As the side length of intersecting squares is at least $\ell_j^i/3$ (see Lemma 6.9) these points suffice to find all intersecting squares. From the remaining squares the algorithm iteratively chooses the smallest one. By querying τ_i with the four corner points of the chosen square we identify and remove all intersecting squares, which are larger and thus

must contain one of the corner points. Since a deletion takes $O(\log^2 n)$ time, Algorithm 6.2 needs $O(n \log^2 n)$ time per level. The storage of τ_i is $O(n \log n)$.

Since there are $O(\log n)$ levels the total running time is $O(n \log^3 n)$. □

Congruent Axis-Aligned Square Cones

For axis-aligned congruent square cones, all squares E_j^i in plane π_i have the same size. This has two consequences. First, it simplifies implementing Algorithm 6.2: any square is "smallest". Second, at most four independent squares can intersect a given square. Thus the analysis in the previous section immediately yields a (1/8)-approximation. We can however, do better in this case by using a denser set of planes, that is by setting the parameter k to a larger value.

Recall that $\alpha = (k+1)/k$. We define π_i to be the horizontal plane at scale $\sigma_i = S_{\max}/\alpha^i$ for $i = 0, \ldots, N_k$. Note that for $k = 1$ this yields the same set of planes as in the case of arbitrary square cones that we considered above. As before, $\mathcal{S}_i$ and $\mathcal{A}_i$ denote the active segments of the extrusions between π_{i-1} and π_i in the optimal solution and the solution of our algorithm, respectively.

Lemma 6.11 *For each $i \in \{1, 2, \ldots, N_k - 1\}$ it holds that $H(\mathcal{A}_{i+1}) \geq H(\mathcal{S}_i)/(4\alpha)$.*

Proof. Let E_j^i be a square in π_i that is active in $\mathcal{A}$. Since all squares in π_i have the same size there are at most four squares intersecting E_j^i that are active in $\mathcal{S}$; let D_j^i denote the set of those squares. Since the height difference between π_{i-1} and π_i is α times the height difference between π_i and π_{i+1}, we know that the contribution of E_j to $H(\mathcal{A}_{i+1})$ is at least $1/(4\alpha)$ times the contribution of D_j^i to $H(\mathcal{S}_i)$. □

Theorem 6.14 *Algorithm 6.2 computes a $(1/4 - \varepsilon)$-approximation for the maximum total active range height of a set of n axis-aligned congruent square cones in $O(n/\varepsilon \cdot \log(n/\varepsilon) \log n)$ time and $O(n \log n)$ space.*

Proof. Given Lemma 6.11 it remains to compare $H(\mathcal{S}_{N_k}) + H(\mathcal{S}_{N_k+1})$ to $H(\mathcal{A}_{N_k+1}) + H(\mathcal{A}_1)$. The scale of π_{N_k-1} is at most $\alpha \cdot S_{\max}/(kn)$ and obviously there are at most n active cone segments in $\mathcal{S}$ below π_{N_k-1}, so their total active range height is at most $\alpha/k \cdot S_{\max}$. On the other hand, there is at least one active cone segment in $\mathcal{A}_1$ of height $S_{\max}/(k+1)$. We can charge $H(\mathcal{S}_{N_k}) + H(\mathcal{S}_{N_k+1})$ to $H(\mathcal{A}_1)$, which is smaller by at most a factor of $1/\alpha^2 \geq 1/4$. Together with Lemma 6.11 this implies an approximation factor of $1/(4\alpha)$. Let $k = 1/(4\varepsilon) - 1$ to obtain a $(1/4 - \varepsilon)$-approximation.

The implementation of Algorithm 6.2 uses an augmented dynamic range tree [MN90] for each level. It stores the squares E_j^i, $j = 1, \ldots, n$, in the plane π_i represented as the apexes of their underlying cones in the (x, y)-plane. This is possible since all cones are congruent and thus the mapping between apexes and squares is bijective. A query for all squares that intersect a given square E_j^i easily translates into a range query for the corresponding apexes in the range tree. The augmented dynamic range tree can be constructed in $O(n \log n)$ time and space since only deletions and no insertions need to be supported. Each deletion takes $O(\log n)$ time such that the running time per level is $O(n \log n)$ (see the proof of Theorem 6.13). The number of levels in Algorithm 6.2 is $O(\log_\alpha kn) = O((\log kn)/\log \alpha)$. For $k \geq 1$ we have $\alpha^k \geq 2$ and thus $1/\log \alpha \leq k$. Now we can bound the number of levels by $O(k \log(kn)) = O(1/\varepsilon \cdot \log(n/\varepsilon))$. This yields the desired running time of $O(n/\varepsilon \cdot \log(n/\varepsilon) \log n)$. The space consumption remains $O(n \log n)$. □

6.4 Concluding Remarks

Dynamic map labeling is an exciting new subdiscipline of map labeling inspired by interactive web-mapping applications. In this chapter we have presented an extensive, rigorous algorithmic study of the active range optimization (ARO) problem for dynamic map labeling. For labeling features in the 2d-plane we have shown that even the simplest variants of ARO are NP-complete. For 1d-ARO we have shown NP-completeness for a variant where the extrusions are squares of at least two different sizes. In the algorithmic part we have presented a toolbox of an exact algorithm and several approximation algorithms for 1d- and 2d-variants of ARO with various extrusion shapes, see Table 6.1 in Section 6.1.

Open problems. In spite of the multitude of results obtained in this chapter, many unsolved open questions remain in dynamic map labeling. An obvious question is whether any approximation factor can be improved, or whether any of the problems admits a polynomial-time approximation scheme. (Some obvious attempts for such a scheme do not work, see Section 6.3.3.) Furthermore, the complexity of general 1d-ARO is still unknown for regular shapes such as unit squares, congruent trapezoids, and segments of congruent triangles.

Mapping applications in practice often need to work with more complex labeling models, such as labels of different lengths and fonts; non-axis-aligned labels; non-rectangular labels, such as a road label that follows a curvy road; and sliding labels—that is, non-invariant point placements. Any of these raises a number of interesting theoretical questions. Moreover, given the growing popularity of interactive dynamic maps, it is important to devise efficient heuristics or approximation algorithms that can deal with these more realistic labeling models and that at the same time adhere to the consistency criteria in order to resolve the labeling inconsistencies found in today's systems.

Chapter 7

Optimal Tanglegram Layout

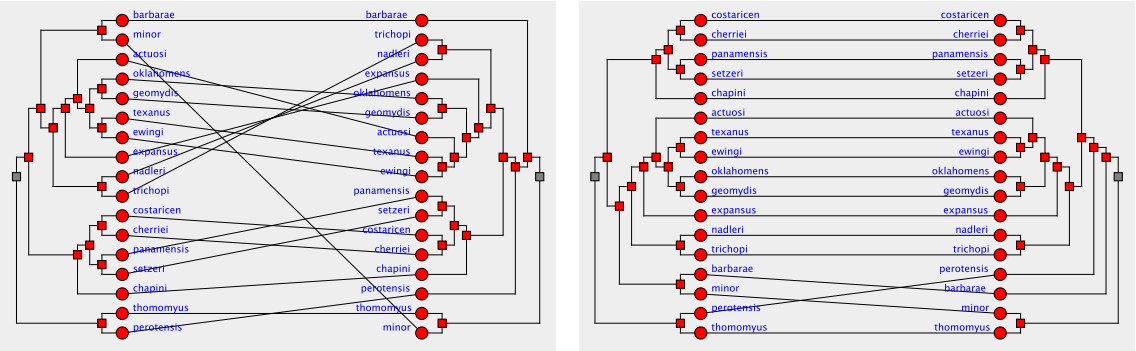
A *tanglegram* is a comparative visualization of a pair of trees on the same set of leaves. It consists of two facing planar tree drawings such that the leaves are arranged on two parallel lines, one for each tree. Since both trees use the same set of leaves we can connect each pair of corresponding leaves in the two trees by a straight-line *inter-tree edge*. Edge crossings are a major factor reducing the readability of graph drawings [Pur97] and thus a natural objective for a tanglegram is to find plane drawings of the two trees such that the number of inter-tree edge crossings is minimized. Applications of tanglegrams arise in areas that deal with comparing visualizations of hierarchical data in the form of trees, for example, phylogenetics or clustering.

In this chapter we study the tanglegram layout problem theoretically and experimentally with a focus on binary trees. On the theoretical side, we show that minimizing the number of crossings is NP-complete, even for complete binary trees. On the one hand, we show that under the Unique-Games Conjecture the problem is hard to approximate within a constant factor for *general* binary trees, but, on the other hand, we give a recursive factor-2 approximation algorithm for *complete* binary trees. For the latter case we also give a simple fixed-parameter algorithm. On the experimental side, we are interested in algorithms that can solve instances as they typically arise in the application areas optimally or at least near-optimally. To that end we give an integer-linear program and a new branch-and-bound algorithm to compute exact solutions on which we base our experimental analysis. The experiments evaluate the performance of several algorithms, including our recursive algorithm, previously suggested heuristics, and a new greedy heuristic that is based on our exact branch-and-bound algorithm. It turns out that the greedy heuristic is at least as good as the best of the other algorithms and often finds optimal solutions; at the same time it is fast enough to be applied in practice. The chapter is based on joint work with Kevin Buchin, Maike Buchin, Jaroslaw Byrka, Danny Holten, Yoshio Okamoto, Rodrigo I. Silveira, Markus Völker, and Alexander Wolff [BBB+09, NVWH09].

7.1 Introduction

Tanglegrams are visualizations of pairs of trees whose leaf sets are in one-to-one correspondence [Pag02]. The need to visually compare pairs of trees arises in applications such as evolutionary biology or clustering.

In biology, a phylogenetic tree (or *phylogeny*) is a (rooted) binary tree that describes a hypothesis of the evolutionary history of a set of species (or *taxa*) that are placed at



(a) Arbitrary layout. (b) Optimal layout.

Figure 7.1: A binary tanglegram of two phylogenies for lice of pocket gophers [HSV+94].

the leaves of the tree. Each inner node represents a potential ancestor of the taxa at its child nodes and the general aim is to group together related species into subtrees. Tree reconstruction is usually done based on the DNA or protein sequences of a set of taxa according to various criteria and by various algorithms such as maximum parsimony, maximum likelihood, or neighbor joining [NK00], to name but a few. Since none of these methods can guarantee to reconstruct the correct tree, biologists often generate a set of candidate trees. For them it is important to have good support for comparing trees in order to draw the right conclusions from potentially contradicting tree topologies. Different phylogenies can be compared using various tree-distance measures [DHJ+97, Bil05], but it is also important to graphically depict the trees. A good drawing of the trees helps biologists to recognize and compare different substructures of the phylogenies. It provides more information than a single numeric value such as a tree distance can do. Moreover, tree drawings are often used to communicate the findings of a study to its readers. For this purpose, tanglegrams are now a frequently used visualization method for pairs of related phylogenies [Pag02]. Figure 7.1 shows an example.

In clustering, our second application, clusters are often computed incrementally: in the beginning each object forms its own cluster, and then, step by step, the pair of clusters that is closest according to some distance measure is joined. This is called *agglomerative* clustering. Alternatively, in *divisive* clustering, there is one initial cluster containing all objects, which is iteratively split into two child clusters until we end up with the set of singleton clusters. Such a hierarchical clustering is naturally represented by a binary tree called *dendrogram*, where elements are represented by the leaves and each inner node of the tree represents a cluster containing the leaves in its subtree. Similarly to phylogenetic tree reconstruction, a dendrogram often reflects only a hypothesis of how the clustered objects could be related. Again, there are various clustering algorithms and parameter settings to generate multiple different dendrograms for the same data. From similarities and differences found in a set of dendrograms a data analyst can gain valuable insights into the nature of the studied data. Thus tanglegrams can be used as a means to visually compare pairs of dendrograms in clustering applications just as they are used in phylogenetics.

From the application point of view it makes sense to insist that (a) the two trees themselves are drawn plane, that is, without edge crossings, (b) each leaf of one tree is connected by an *inter-tree* edge to the corresponding leaf in the other tree, and (c) the number of crossings among the inter-tree edges is minimized. This results in legible

tanglegram drawings: each tree is drawn in the usual style (for example rectilinear) and distracting crossings of inter-tree edges are kept to a minimum. Edge crossings have been empirically found as a major source of reduced legibility of graph layouts [PCJ97, Pur97]. Note that we are interested in the minimum number of crossings for visualization purposes. The number of crossings is not intended to be a tree-distance measure. Examples for such measures are nearest-neighbor interchange, subtree transfer [DHJ+97], and tree edit distance [Bil05]. Following the bioinformatics literature [Pag02, LPR+08], we refer to this crossing minimization problem as the *tanglegram layout* problem; Fernau et al. [FKP05] used the term *two-tree crossing minimization*.

Model. We denote the set of leaves of a tree T by $L(T)$. We are given two rooted trees S and T with n leaves each. The trees S and T must be *uniquely leaf-labeled*, that is, there are bijections $\lambda_S : L(S) \to \Lambda$ and $\lambda_T : L(T) \to \Lambda$ into the set of labels Λ, for example, $\Lambda = \{1, 2, \ldots, n\}$. These two labelings induce the set of inter-tree edges $E_{ST} = \{uv \mid u \in L(S), v \in L(T), \lambda_S(u) = \lambda_T(v)\}$. Now the tanglegram layout problem can be stated formally as follows.

Problem 7.1 (Tanglegram Layout (TL)) *Given a pair $\langle S, T\rangle$ of two uniquely leaf-labeled rooted trees S and T on n leaves, find a* tanglegram layout*, that is, two plane drawings of S and T, such that*

1) the drawing of S is to the left of the line $x = 0$ with all leaves on $x = 0$,
2) the drawing of T is to the right of the line $x = 1$ with all leaves on $x = 1$,
3) the inter-tree edges E_{ST} are drawn as straight-line segments, and
4) the number of inter-tree edge crossings is minimum.

Let the *crossing number* of a TL instance $\langle S, T\rangle$ be the minimum number of inter-tree edge crossings of any tanglegram layout of $\langle S, T\rangle$. Given a tree T, we say that a linear order of $L(T)$ is *compatible* with T if for each node v of T the leaves in the subtree of v form an interval in the order. Note that the TL problem is purely combinatorial. Given a permutation π of $\{1, \ldots, n\}$, we call (i, j) an *inversion* in π if $i < j$ and $\pi(i) > \pi(j)$. For fixed orders σ of $L(S)$ and τ of $L(T)$ we define the permutation $\pi_{\tau,\sigma}$, which for a given position in τ returns the position in σ of the leaf having the same label. Now in this terminology, the TL problem consists of finding an order σ of $L(S)$ compatible with S and an order τ of $L(T)$ compatible with T such that the number of inversions in $\pi_{\tau,\sigma}$ is minimum.

Contributions. Our focus is on binary tanglegrams if not stated otherwise. We mention the extensibility of our results to more general trees where appropriate; still, binary trees are most relevant in phylogenetics and clustering.

There are multiple contributions of this chapter. First, in Section 7.3, we show that binary TL is essentially as hard as the MinUncut problem. If the (widely accepted) Unique Games Conjecture holds, it is NP-hard to approximate MinUncut—and thus binary TL—within any constant factor [KV05]. This motivates us to consider tanglegrams for *complete* binary trees. It turns out that this special case has a rich structure. We start our investigation by giving a reduction from Max2-Sat that establishes the NP-hardness of complete binary TL.

In Section 7.4, the main part of this chapter, we present several algorithmic approaches to the TL problem, both for complete and for general binary trees. We start with a recursive factor-2 approximation algorithm for complete binary TL that runs in cubic time. Our algorithm can also process general binary tanglegrams—without guaranteeing any

approximation ratio. For the dual problem that maximizes the number of non-crossing edges we give a reduction to MaxCut for which the algorithm of Goemans and Williamson yields a 0.878-approximation [GW95]. This result applies to general binary trees. Another way to look at complete binary TL is by studying its parameterized complexity. We give a fixed-parameter algorithm that runs in $O(4^k n^2)$ time, where k is the crossing number of the given instance. The algorithm is much faster and simpler than the previous fixed-parameter algorithm by Fernau et al. for general binary trees [FKP05]. An interesting feature of our algorithm is that the parameter does *not* drop in each level of the recursion. Finally, we state two methods to compute optimal solutions for binary TL: a simple integer linear program (ILP) and a new branch-and-bound algorithm. The branch-and-bound algorithm has a variable for the order of the children in each node of the tanglegram. The algorithm exploits the fact that the variables can be chosen independently; it chooses a variable ordering that often yields a very good first solution, whose value can then be used as upper bound to prune many branches of the search tree. The algorithm takes exponential time in general, but yields a fast and simple, yet effective greedy heuristic. The heuristic outputs—in quadratic time—the first solution that the branch-and-bound algorithm finds.

Finally, we perform an extensive experimental comparison of our recursive algorithm, our new heuristic, and two previously known heuristics for binary TL. We present the results of the comparison in Section 7.5. We measure the performance of the heuristics with respect to the optimum, which we compute with the above exact methods. The test data comprise small and medium-sized tanglegrams of up to a few hundred leaves, which are predominantly found in practical applications. They contain randomly generated complete and general binary trees as well as a large collection of real-world phylogenies. It turns out in the experiments that our greedy heuristic is both very effective in finding near-optimal solutions and fast enough to be useful in practice.

7.2 Related Work

In graph drawing the so-called *two-sided crossing minimization problem* (2SCM) is an important problem that occurs when computing layered graph layouts. Such layouts have been introduced by Sugiyama et al. [STT81] and are widely used for drawing hierarchical graphs. In 2SCM, vertices of a bipartite graph are to be placed on two parallel lines (called *layers*) such that for each vertex on one line all its adjacent vertices lie on the other line. As in TL the objective is to minimize the number of edge crossings provided that edges are drawn as straight-line segments. In one-sided crossing minimization (1SCM) the order of the vertices on one of the layers is fixed. Already 1SCM is NP-hard [EW94], even if the given graph is a forest of 4-stars [MUV02]. In contrast to TL, a vertex in an instance of 1SCM or 2SCM can have multiple incident edges and the linear order of the vertices is not restricted to be compatible with the two input trees.

The following is known about 1SCM in terms of approximation and exact algorithms. The median heuristic of Eades and Wormald [EW94] yields a 3-approximation and a randomized algorithm of Nagamochi [Nag05] yields an expected 1.4664-approximation. Dujmovič et al. [DFK04] gave a fixed-parameter algorithm that runs in $O^\star(1.4664^k)$ time, where k is the minimum number of crossings in any 2-layer drawing of the given graph that respects the vertex order of the fixed layer. The $O^\star(\cdot)$-notation ignores polynomial factors.

Jünger and Mutzel [JM97] performed an experimental comparison of exact and heuristic algorithms for both 1SCM and 2SCM. Their main findings were that for 1SCM the exact

solution can be computed quickly for up to 60 vertices in the free layer, and for 2SCM an iterated barycenter heuristic is the method of choice for instances with more than 15 vertices in each layer.

Dwyer and Schreiber [DS04] studied drawing a series of tanglegrams in 2.5 dimensions, that is, the trees are drawn on a set of stacked two-dimensional planes. They considered a one-sided version of binary TL by fixing the layout of the first tree in the stack, and then, layer-by-layer, computing the optimal leaf order of the next tree with respect to the previous one in $O(n^2 \log n)$ time each. Such a one-sided TL problem is denoted as *one-tree crossing minimization* (1TCM). We include an iterated version of the 1TCM algorithm that alternately fixes one of the trees and optimizes the other as a heuristic in our experimental comparison. Note that the efficient algorithm of Dwyer and Schreiber for 1TCM contrasts the NP-hardness of 1SCM.

Fernau et al. [FKP05] showed how to solve 1TCM in $O(n \log^2 n)$ time, proved that binary TL is NP-hard, and gave a fixed-parameter algorithm that runs in $O^\star(c^k)$ time, where c is a constant that Fernau et al. estimate to be 1024, and k is the crossing number of the given tanglegram. They also made the simple observation that the edges of the tanglegram can be directed from one root to the other. Thus the existence of a planar tanglegram drawing can be verified using a linear-time upward-planarity test for single-source directed acyclic graphs [BdBMT98]. Later, apparently unaware of these results, Lozano et al. [LPR$^+$08] gave a quadratic-time algorithm for the same special case, to which they refer as *planar tanglegram layout*.

Zainon and Calder [ZC06] described an interactive tree comparison tool that allows manual and automatic rearrangement of a tanglegram. Their general aim was to highlight differences and similarities in the two trees; minimizing the number of inter-tree edge crossings is not a direct goal of their methods. Two heuristics were implemented. The first heuristic starts at the roots of both trees and flips the subtrees of one tree if this increases the number of edges between the aligned subtrees; then it recurses on both pairs of aligned subtrees. The second heuristic minimizes the *triplet difference* between two n-leaf trees over all 2^{2n-2} possible arrangements. The triplet difference counts the number of all three-leaf subsets for which the respective induced subtrees differ in the two trees. They recommended the following semi-automatic approach: use the first algorithm to find a layout of the full trees and then untangle small groups of edges individually using the second (exponential-time) algorithm, followed by some manual fine tuning.

Holten and van Wijk [HvW08] presented a tanglegram visualization tool for the comparison of pairs of large (not necessarily binary) trees. Their tool repeatedly applies the barycenter method [STT81] to reduce inter-tree crossings and a subsequent edge-bundling technique to reduce visual clutter. The crossing reduction heuristic of Holten and van Wijk is included in our experimental evaluation.

7.3 Complexity

In this section we consider the complexity of binary TL, which Fernau et al. [FKP05] have shown to be NP-complete for general binary tanglegrams. We strengthen their findings in two ways. First, we show that it is unlikely that an efficient constant-factor approximation for general binary TL exists. Second, we show that TL remains hard even when restricted to *complete* binary tanglegrams.

We start by showing that binary TL is essentially as hard as the MinUncut problem.

This relates the existence of a constant-factor approximation for TL to the Unique Games Conjecture (UGC). The UGC was introduced by Khot [Kho02] in the context of interactive proofs. It concerns a scenario with two provers and a single round of answers to a question of the verifier. The word "unique" refers to the strategy of the verifier, who for any fixed answer of one of the provers will accept the proof only if the other prover gives the unique second part of the proof. The provers cannot communicate with each other. Still they want to maximize the probability of the proof being accepted given that questions of the verifier are drawn randomly from a given distribution. The UGC conjectures that it is NP-hard to decide whether the optimal strategy of the provers gives them a high probability of success.

The UGC became famous when it was discovered that it implies optimal hardness-of-approximation results for problems such as MaxCut and VertexCover, and forbids constant factor-approximation algorithms for problems such as MinUncut and SparsestCut [KV05]. We reduce the MinUncut problem to the TL problem, which, by the result of Khot and Vishnoi [KV05], makes it unlikely that an efficient constant-factor approximation for TL exists.

The MinUncut problem is defined as follows. Given an undirected graph $G = (V, E)$, find a partition (V_1, V_2) of the vertex set V that minimizes the number of edges that are not cut by the partition, that is, $\min_{(V_1,V_2)} |\{uv \in E : u, v \in V_1 \text{ or } u, v \in V_2\}|$. Note that computing an optimal solution to MinUncut is equivalent to computing an optimal solution to MaxCut. Nevertheless, the MinUncut problem is more difficult to approximate.

Theorem 7.1 *Under the Unique Games Conjecture it is NP-hard to approximate the TL problem (Problem 7.1) for general binary trees within any constant factor.*

Proof. As mentioned above we reduce from the MinUncut problem. Our reduction is similar to the reduction in the NP-hardness proof by Fernau et al. [FKP05].

Consider an instance $G = (V, E)$ of the MinUncut problem. We construct a TL instance $\langle S, T\rangle$ as follows. The two trees S and T are identical and there are three groups of edges connecting leaves of S to leaves of T. For simplicity we define multiple edges between a pair of leaves. In the actual trees we can replace each such leaf by a binary tree with the appropriate number of leaves.

Suppose $V = \{v_1, v_2, \ldots, v_n\}$, then both S and T are constructed as follows. There is a *backbone* path $(v_1^1, v_1^2, v_2^1, v_2^2, \ldots, v_n^1, v_n^2, a)$ from the root node v_1^1 to a leaf a. Additionally, there are leaves $l_S(v_i^j)$ and $l_T(v_i^j)$ attached to each node v_i^j for $i \in \{1, \ldots, n\}$ and $j \in \{1, 2\}$ in S and T, respectively. The construction of S and T is illustrated in Figure 7.2 for the complete graph $K_3 = (\{v_1, v_2, v_3\}, \{v_1v_2, v_2v_3, v_3v_1\})$. The edges of S and T form the following three groups:

- Group A contains n^{11} edges connecting $l_S(a)$ with $l_T(a)$.
- Group B contains for every $v_i \in V$ n^7 edges connecting $l_S(v_i^1)$ with $l_T(v_i^2)$, and n^7 edges connecting $l_S(v_i^2)$ with $l_T(v_i^1)$.
- Group C contains for every $v_iv_j \in E$ a single edge from $l_S(v_i^1)$ to $l_T(v_j^1)$.

Next we show how to transform an optimal solution of the MinUncut instance into a solution of the corresponding TL instance. Suppose that in the optimal partition (V_1^*, V_2^*) of G there are k edges that are not cut. Then we claim that there exists a drawing of $\langle S, T\rangle$ such that $k \cdot n^{11} + O(n^{10})$ pairs of edges cross. In the example of Figure 7.2 we consider the cut $(\{v_1\}, \{v_2, v_3\})$ with the uncut edge v_2v_3. It suffices to draw, for each vertex $v_i \in V_1^*$

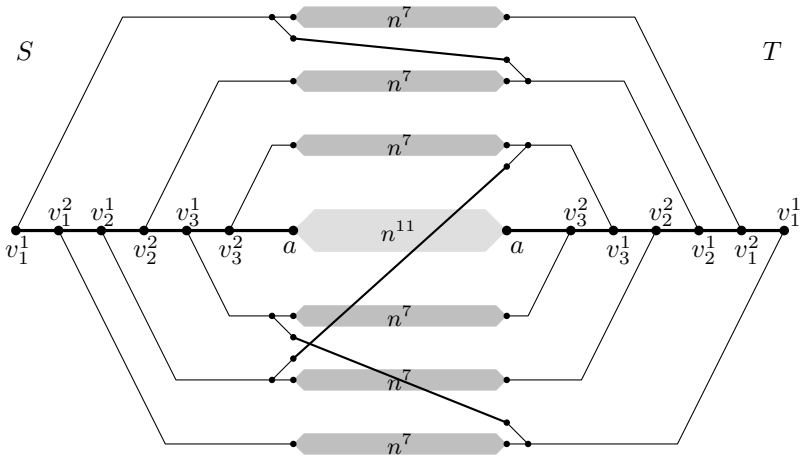


Figure 7.2: TL instance corresponding to the graph K_3 and the cut $(\{v_1\}, \{v_2, v_3\})$.

($v_i \in V_2^*$), the leaves $l_S(v_i^1)$ and $l_T(v_i^2)$ above (below) the backbones, and the nodes $l_S(v_i^2)$ and $l_T(v_i^1)$ below (above) the backbones. It remains to count the crossings: there are $k \cdot n^{11}$ A–C crossings, no A–B crossings, $O(n^{10})$ B–C crossings, and $O(n^4)$ C–C crossings. In Figure 7.2, we have $k = 1$ and accordingly $n^{11} + 2n^7 + 1$ crossings in total.

Now suppose there exists an α-approximation algorithm for the TL problem with some constant α. Applying this algorithm to the instance $\langle S, T \rangle$ defined above yields a drawing $D(S, T)$ with at most $\alpha \cdot k \cdot n^{11} + O(n^{10})$ crossings. Let us assume that n is much larger than α. We show that from such a drawing $D(S, T)$ we would be able to reconstruct a cut (V_1, V_2) in G with at most $\alpha \cdot k$ edges uncut. First, observe that if a node $l_S(v_i^1)$ is drawn above (below) the backbone in $D(S, T)$, then $l_T(v_i^2)$ must be drawn on the same side of the backbone, otherwise it would result in n^{18} A–B crossings. Similarly $l_S(v_i^2)$ must be on the same side as $l_T(v_i^1)$. Then observe that if a node $l_S(v_i^1)$ is drawn above (below) the backbone in $D(S, T)$, then $l_S(v_i^2)$ must be drawn below (above) the backbone, otherwise there would be $O(n^{14})$ B–B crossings. Finally, observe that if we interpret the set of vertices v_i for which $l_S(v_i^1)$ is drawn above the backbone as a set V_1 of a partition of G, then this partition leaves at most $\alpha \cdot k$ edges from E uncut.

Hence, an α-approximation for the TL problem provides an α-approximation for the MinUncut problem, which contradicts the UGC. □

The above negative result for (general) binary TL is our motivation to investigate the complexity of complete binary TL. It turns out that even this special case is hard. Unlike Fernau et al. [FKP05] who show hardness of binary TL by a reduction from MaxCut using extremely unbalanced trees, we use a quite different reduction from a variant of Max2-Sat.

Theorem 7.2 *The TL problem (Problem 7.1) is NP-complete even for complete binary tanglegrams.*

Proof. The Max2-Sat problem is defined as follows. Given a set $U = \{x_1, \ldots, x_n\}$ of Boolean variables, a set $C = \{c_1, \ldots, c_m\}$ of disjunctive clauses containing two literals each, and an integer K, the question is whether there is a truth assignment of the variables such that at least K clauses are satisfied. We consider a restricted version of Max2-Sat, where each variable appears in at most three clauses. This version remains NP-complete [RRR98].

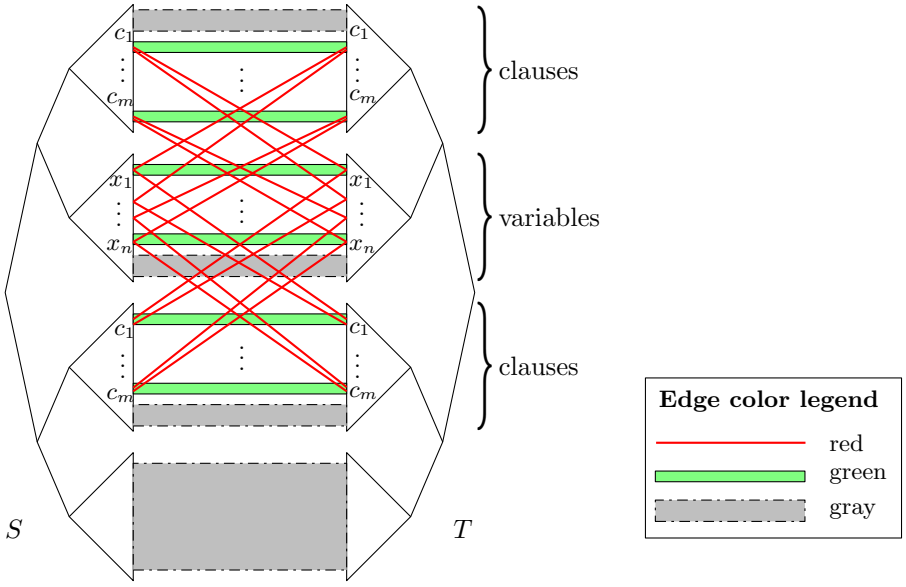


Figure 7.3: High-level structure of the two trees S and T. Red edges connect clause and variable gadgets, green edges connect corresponding gadget halves, and gray edges are dummy edges to complete the trees.

Our reduction constructs two complete binary trees S and T, in which certain aligned subtrees serve as variable gadgets and others as clause gadgets. We further determine an integer K' such that the instance $\langle S, T\rangle$ has less than K' crossings if and only if the corresponding Max2-Sat instance has a truth assignment that satisfies at least K clauses.

The high-level structure of the two trees is depicted in Figure 7.3. From top to bottom, the four subtrees at level 2 on both sides are a clause subtree, a variable subtree, another clause subtree, and finally a dummy subtree. The subtrees are connected to each other by inter-tree edges such that in any optimal solution they must be aligned in the depicted (or mirrored) order. Each clause gadget appears twice, once in each clause subtree, and is connected to the variable gadgets belonging to its two literals. Pairs of corresponding gadgets in S and T are connected to each other. Finally, non-crossing dummy edges connect unused leaves in order to make S and T complete. In the following we describe the gadgets in more detail.

Variable gadgets. The basic structure of a variable gadget consists of two complete binary trees with 32 leaves each as shown in Figure 7.4. Each tree has three highlighted subtrees of size 2 labeled a, b, c and a', b', c', respectively. From each of these subtrees there is one red *connector* edge leaving the gadget at the top and one leaving it at the bottom. As long as two connector edges from the same tree do not cross each other, they transfer the vertical order of the labeled subtrees towards a clause gadget. We define the configuration in Figure 7.4a as *true* and the configuration in Figure 7.4b as *false*. If the configuration is in its *true* state, the induced vertical order of the connector edges is $a < b < c$, otherwise the order is inverse: $c < b < a$. It can easily be verified that both states have the same number of crossings. To see that it is optimal observe that each pair of connector edges from the same subtree (for example, subtree a) always crosses all 26 gray edges in the gadget. Furthermore all 24 crossings of two connector edges in the figure are mandatory. Finally, the four crossings among the gray edges between subtrees 1 and $2'$ and subtrees 2 and $1'$ are also optimal. (Otherwise, if subtree 1 is aligned with subtree $2'$, there are at

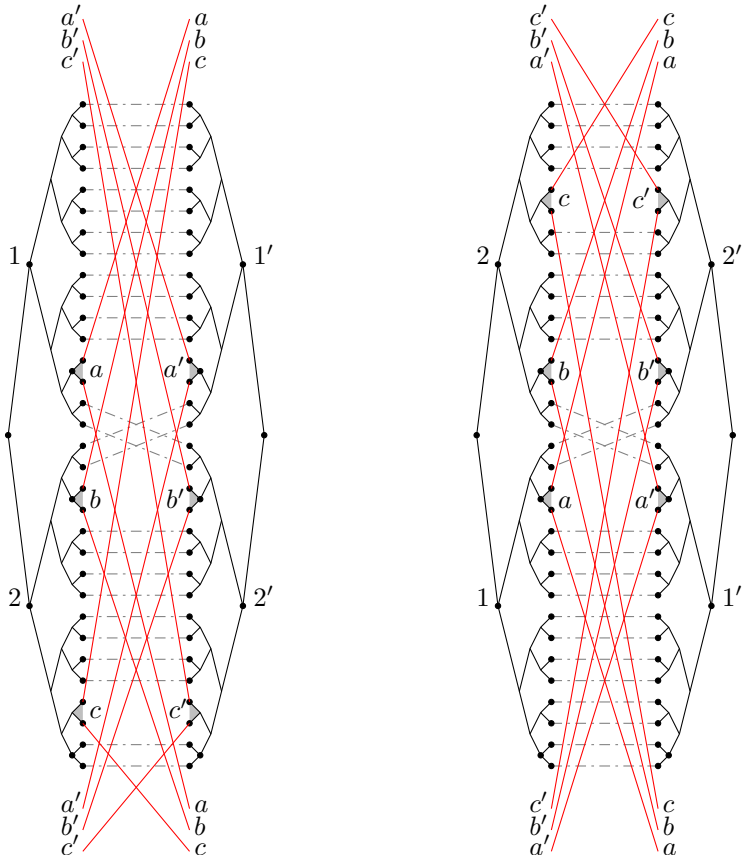


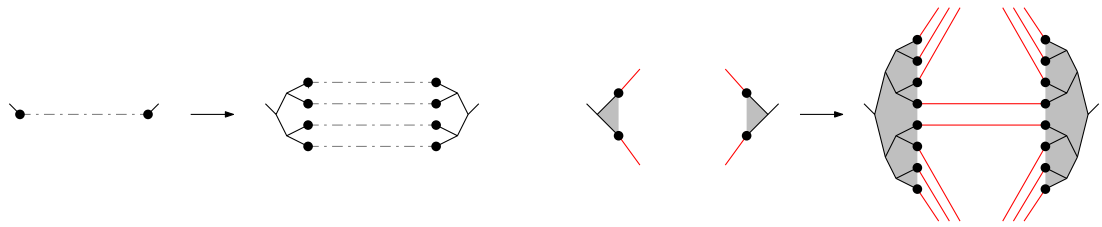
Figure 7.4: The variable gadget in its two optimal configurations with 184 crossings. Red edges are drawn solid, whereas dash-dot style is used for gray edges.

least 120 gray–gray crossings in addition to the 24 red–red crossings and the 156 red–gray crossings as opposed to a total of 184 crossings in either configuration of Figure 7.4.)

Note that so far the gadget in the figure is designed for a single appearance of the variable since the four connector-edge triplets are required for a single clause. For the Max2-Sat reduction, however, each variable can appear up to three times in different clauses. By appending a complete binary tree with four leaves as in Figure 7.5 to each leaf of the gadget in Figure 7.4 and copying each edge accordingly the above arguments still hold for the enlarged trees with 128 leaves each. Unused connector edges in opposite subtrees are linked to each other (a to a' etc.) as in Figure 7.5b such that the number of crossings in the gadget remains balanced for both states.

Clause gadgets. For each clause $c_i = l_{i1} \vee l_{i2}$, where l_{i1} and l_{i2} denote the two literals, we create two clause gadgets: one in the upper clause subtrees and one in the lower clause subtrees (recall Figure 7.3). Each gadget itself consists of two parts: one part that uses the connectors from the first variable in the left tree and those from the second variable in the right tree and vice versa. Figure 7.6 shows one such part of the gadget in the lower clause subtrees, where the connector edges lead upwards. The gadget in the upper clause subtree is simply a mirrored version.

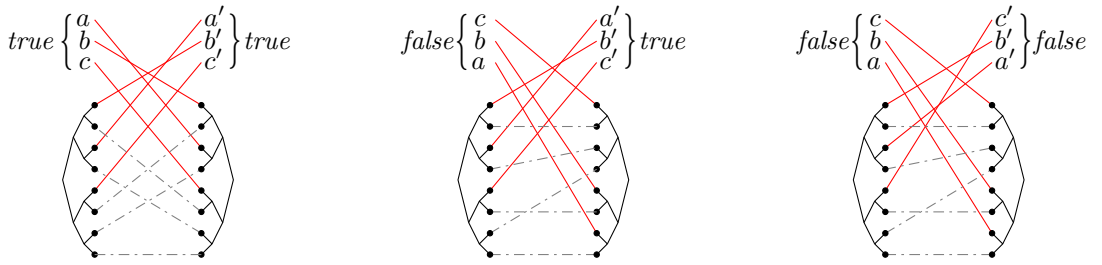
The basic structure consists of two aligned subtrees with eight leaves as depicted in



(a) A single gray edge.

(b) Two pairs of connector edges for a variable used in three clauses.

Figure 7.5: Replacing each edge by four edges.



(a) *true* ∨ *true*: 21 crossings.

(b) *false* ∨ *true*: 21 crossings.

(c) *false* ∨ *false*: 22 crossings.

Figure 7.6: The clause gadget for a clause $c_i = l_{i1} \vee l_{i2}$.

Figure 7.6. Three of the leaves on each side serve as the missing endpoints for the triplets of connector edges from the corresponding variables. Recall that for a positive literal with value *true* the order of the connector edges is $a < b < c$, and for a positive literal with value *false* it is $c < b < a$. (For negative literals the meaning of the orders is inverted.) The two connector leaves for the edges labeled a and b are in the same four-leaf subtree, the connector leaf for c is in the other subtree. Three cases need to be distinguished. If (1) both literals are *true*, then the configuration in Figure 7.6a is optimal with 21 crossings. If (2) only one literal is *true*, then Figure 7.6b shows again an optimal configuration with 21 crossings. Here the tree on the right side is rotated in its root node. Finally, if (3) both literals are *false*, there are at least 22 crossings in the gadget as shown in Figure 7.6c. Since this substructure is repeated four times for each clause we have 84 induced crossings for satisfied clauses and 88 induced crossings for unsatisfied clauses.

Reduction. We construct the gadgets for all variables and clauses and link them together as two trees S and T, which are filled up with dummy leaves and edges such that they become complete binary trees. The general layout is as depicted in Figure 7.3, where each dummy leaf in S is connected to the opposite dummy leaf in T such that there are no crossings among dummy edges. In each of the four main subtrees all dummy edges are consecutive. Thus of all dummy edges only those in the variable subtree have crossings with exactly half the connector edges.

It remains to compute the minimum number M of crossings that are always necessary, even if all clauses are satisfied. Then the Max2-Sat instance has a solution with at least K satisfied clauses if and only if the constructed TL instance has a solution with at most $K' = M + 4(|C| - K)$ crossings. We get the corresponding variable assignment directly from the layout of the variable gadgets.

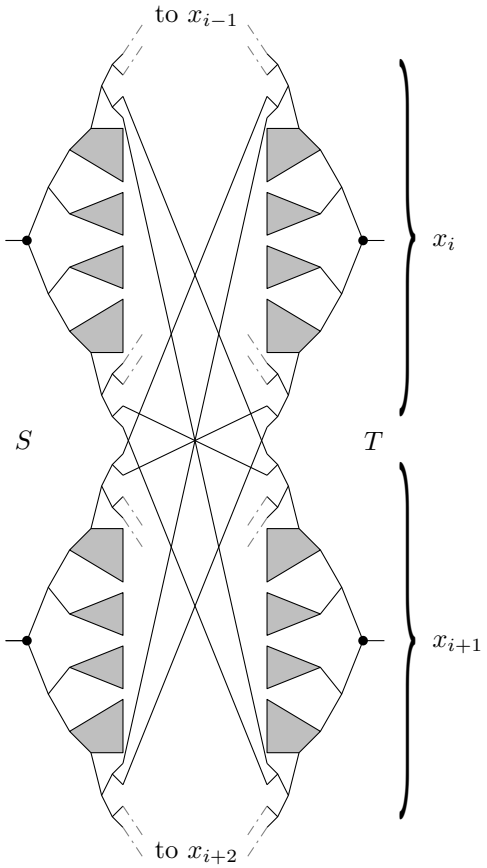


Figure 7.7: Linking adjacent variable gadgets for x_i and x_{i+1}.

The first step for computing M is to fix an order for the variable gadgets in the variable subtree. Let this order be $x_1 < x_2 < \ldots < x_n$. To enforce this as the vertical order of the variable gadgets we need to establish links between adjacent gadgets such that any other order would increase the number of crossings. For these neighbor links we need eight of the 128 leaves in each half of each variable gadget as shown in Figure 7.7. Since both subtrees below the root of x_i in S and both subtrees below the root of x_{i+1} in T are connected to each other, the minimum number of crossings of those edges is independent of the truth state of each gadget. However, separating two adjacent variables by tree rotations at higher levels in S and T leads to a large number of extra crossings since the eight neighbor links would cross all variable gadgets between x_i and x_{i+1}.

Once the order of the variables is fixed, we sort all clauses lexicographically and place smaller clauses towards the top of the clause subtrees. Consider two clause gadgets in the same clause subtree. Then in the given clause order there are crossings between their connector-edge triplets if and only if the intervals between their respective variables intersect in the variable order. Since these crossings are unavoidable, the number of connector-triplet crossings in the lexicographic order of the clauses is optimal. Now we can finally compute all necessary crossings between connector edges, dummy edges and intra-gadget edges which yields the number M.

Since each gadget is of constant size the two trees and the number M can be computed in polynomial time. The fact that the complete binary TL problem belongs to the class $\mathcal{NP}$ follows immediately from the NP-completeness of the general TL problem [FKP05].□

7.4 Algorithms

This section presents our new algorithms for the TL problem. We begin with a 2-approximation algorithm for complete binary TL followed by a reduction of the dual problem TL* that maximizes the number of non-crossing inter-tree edges to MaxCut, which is known to have a 0.878-approximation. After the approximation results we give a fixed-parameter algorithm for complete binary TL. We close the section with two exact algorithms for general binary TL: a simple ILP and a branch-and-bound algorithm that has an exponential worst-case running time but at the same time gives rise to a fast yet effective heuristic.

7.4.1 Approximation Algorithm

We start with a basic observation about binary tanglegrams. As we have noted in the introduction, TL is a purely combinatorial problem, that is, it suffices to determine two leaf orders σ and τ that are compatible with the input trees S and T, respectively. These orders are completely determined by fixing an order of the two subtrees of each inner node $v \in S^\circ \cup T^\circ$, where S° and T° denote the set of inner nodes of S and T. Our algorithm will recursively split the two trees S and T at their roots into two equally sized subinstances and determine leaf orders of S and T by choosing a locally optimal order of the subtrees below the left and right root of the current subinstance.

Let $\langle S_0, T_0 \rangle$ denote the input instance. We assume that an initial layout of S_0 and T_0 is given, that is, the subtrees of each $v \in S_0^\circ \cup T_0^\circ$ are ordered (otherwise choose an arbitrary initial layout). The root of a tree T is denoted as v_T. For a binary tree T with the two ordered subtrees T_1 and T_2 of v_T we use the notation $T = (T_1, T_2)$. For each subinstance $\langle S, T \rangle$ with $S = (S_1, S_2)$ and $T = (T_1, T_2)$ we need to consider the four configurations $(S_1, S_2) \times (T_1, T_2)$ (initial layout), $(S_2, S_1) \times (T_1, T_2)$ (swap at v_S), $(S_1, S_2) \times (T_2, T_1)$ (swap at v_T), and $(S_2, S_1) \times (T_2, T_1)$ (swap at v_S and v_T). For each configuration we recursively solve two subinstances and then choose the configuration with the minimum number of crossings.

We always split the instance $\langle S, T \rangle$ into an upper and a lower half, that is, the subinstances depend on the swap decision. If we swap both v_S and v_T or none, the two subinstances are $\langle S_1, T_1 \rangle$ and $\langle S_2, T_2 \rangle$; if only one side is swapped, the subinstances are $\langle S_1, T_2 \rangle$ and $\langle S_2, T_1 \rangle$. We solve both subinstances independently. In order to achieve the desired approximation ratio, however, we cannot ignore the swap history of the predecessor nodes of v_T and v_S. This history can be regarded as two bit strings h_S and h_T that represent the *swap* and *no-swap* decisions made at the previous steps of the recursion. Figure 7.8 shows an instance $\langle S, T \rangle$ and its swap history.

The history is used to compute the number of *current-level crossings* of $\langle S, T \rangle$, that is, the number of crossings that are caused by the swap decisions made for the current subinstance. The number of current-level crossings and the recursively computed numbers of crossings of the subinstances determine which of the four configurations of the current instance is the best one. An important observation that is necessary to compute the number of current-level crossings is the following.

Observation 7.1 *For each pair of inter-tree edges ab and cd, $a, c \in L(S)$ and $b, d \in L(T)$, the swap decisions of the lowest common ancestors* $\mathrm{lca}(a, c)$ *and* $\mathrm{lca}(b, d)$ *completely determine whether ab and cd cross or not. Given the order of the subtrees of* $\mathrm{lca}(a, c)$ *swapping or not swapping the subtrees of* $\mathrm{lca}(b, d)$ *(and vice versa) causes or removes the crossing of ab and cd.*

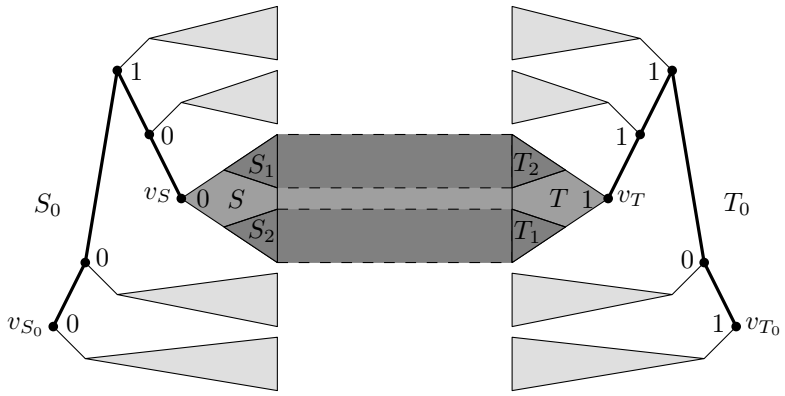


Figure 7.8: The context of an instance $\langle S, T\rangle$ that is split into the subinstances $\langle S_1, T_2\rangle$ and $\langle S_2, T_1\rangle$ since T_1 and T_2 are swapped at v_T. The swap history is indicated by binary swap variables along the paths to the roots v_{S_0} and v_{T_0}.

When considering the current-level crossings of a subinstance $\langle S, T\rangle$ we know from the swap history which of the nodes on the paths P_S and P_T from v_S and v_T to the roots v_{S_0} and v_{T_0} of the full trees, respectively, have swapped their subtrees. Hence for v_S we can compute the current-level crossings of all pairs of edges ab and cd with $a \in L(S_1)$, $c \in L(S_2)$, and $\mathrm{lca}(b, d) \in P_T$; analogously, we can compute the crossings of all pairs of edges ab and cd with $b \in L(T_1)$, $d \in L(T_2)$, and $\mathrm{lca}(a, c) \in P_S$. Note that if $\mathrm{lca}(b, d)$ or $\mathrm{lca}(a, c)$ is not one of the predecessor nodes of v_T or v_S, but it is a node in the subtree T or S, then the crossing of the edges ab and cd will be considered in a subsequent step. Otherwise, our algorithm cannot account for the crossing and we may underestimate the number of crossings. Yet, we are able to bound this error later in Theorem 7.3.

Algorithm 7.1 defines the recursive routine `RecSplit` that computes our tanglegram layout. It is initially called with the parameters $\texttt{RecSplit}(S_0, T_0, \varepsilon, \varepsilon)$, where ε is the empty string.

In order to quickly calculate the number of current-level crossings we use a preprocessing step. To that end we compute two tables $C^=$ and $C^\times$ of size $O(n^2)$. For each pair (v, w) of inner nodes in $S^\circ \times T^\circ$ the entry $C^=[v, w]$ stores the number of crossings of edge pairs ab and cd with $\mathrm{lca}(a, c) = v$ and $\mathrm{lca}(b, d) = w$ if either both or none of v and w swap their subtrees. An entry $C^\times[v, w]$ stores the analogous number of crossings if only one of v and w swap their subtrees.

Lemma 7.1 *The tables $C^=$ and $C^\times$ can be computed in $O(n^2)$ time.*

Proof. We initialize all entries as 0 and preprocess S_0 and T_0 in linear time to support lca-queries in $O(1)$ time [GT83]. Then we determine for each pair of inter-tree edges their lowest common ancestors in S_0 and T_0 and increment the corresponding table entry depending on which two configurations yield the crossing. This takes $O(n^2)$ time for all edge pairs. □

Once we have computed $C^=$ and $C^\times$ we can determine the number of current-level crossings for any subinstance $\langle S, T\rangle$ in $O(\log n)$ time by summing up the appropriate table entries depending on the swap history along the paths P_T and P_S of length $O(\log n)$.

The running time of the algorithm satisfies the recurrence $T(n) \leq 8T(n/2) + O(\log n)$, which solves to $T(n) = O(n^3)$ by the master method [CLRS01]. We now prove that the algorithm yields a 2-approximation.

Algorithm 7.1: `RecSplit` (S, T, h_S, h_T)

Input: n-leaf trees $S = (S_1, S_2)$ and $T = (T_1, T_2)$, swap history h_S and h_T
Output: lower bound cr_{ST} on number of crossings;
orders σ and τ for the leaves of S and T, respectively

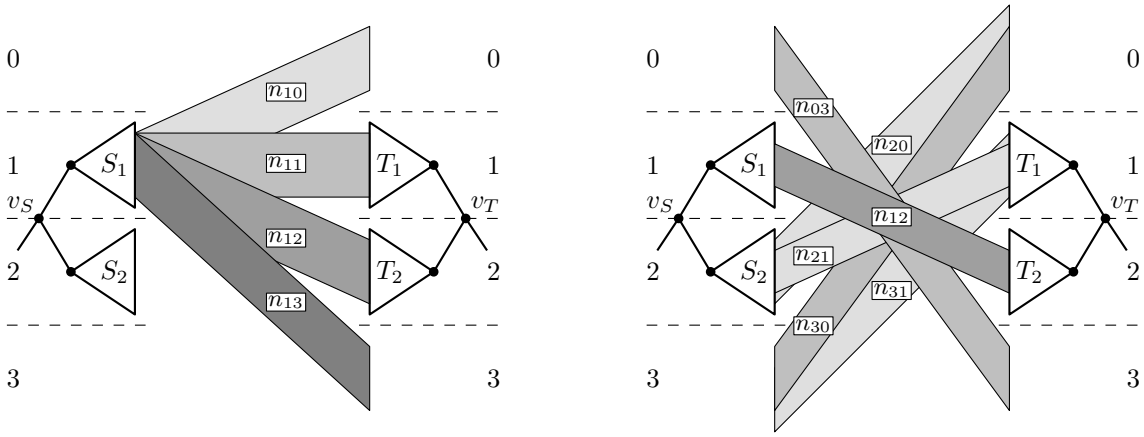
```
if n = 1 then
    return (cr_ST, σ, τ) = (0, v_S, v_T)
else
    cr_ST = ∞
    foreach (swp_S, swp_T) ∈ {0,1}^2 do
        loop through all four cases to swap subtrees of S and T
        cl ← current level crossings induced by (swp_S, swp_T)
        (cr_1, σ_{1+swp_S}, τ_{1+swp_T}) ← RecSplit(S_{1+swp_S}, T_{1+swp_T}, (h_S, swp_S), (h_T, swp_T))
        (cr_2, σ_{2−swp_S}, τ_{2−swp_T}) ← RecSplit(S_{2−swp_S}, T_{2−swp_T}, (h_S, swp_S), (h_T, swp_T))
        if cl + cr_1 + cr_2 < cr_ST then
            cr_ST ← cl + cr_1 + cr_2
            if swp_S = 0 then
                σ ← (σ_1, σ_2)
            else σ ← (σ_2, σ_1)
            if swp_T = 0 then
                τ ← (τ_1, τ_2)
            else τ ← (τ_2, τ_1)
    return (cr_ST, σ, τ)
```

Theorem 7.3 *Given a complete binary tanglegram instance $\langle S_0, T_0\rangle$ with n leaves in each tree, Algorithm 7.1 computes in $O(n^3)$ time a drawing of $\langle S_0, T_0\rangle$ that has at most twice as many crossings as an optimal drawing.*

Proof. Fix any drawing δ of $\langle S_0, T_0\rangle$. The algorithm tries, for each subinstance $\langle S, T\rangle$ of $\langle S_0, T_0\rangle$, all four possible configurations of $S = (S_1, S_2)$ and $T = (T_1, T_2)$—among them the configuration in δ. Assume that the configuration in δ is $\langle (S_1, S_2), (T_1, T_2)\rangle$. We determine an upper bound on the number of crossings that our algorithm fails to count for the drawing δ. In each of the trees S_0 and T_0 we distinguish four different areas for the endpoints of the edges: above S_1, in S_1, in S_2, below S_2 and similarly above T_1, in T_1, in T_2, below T_2. We number these regions from 0 to 3, see Figure 7.9. This allows us to classify the edges into 16 groups (two of which, 0–0 and 3–3, are not relevant). We denote the number of *i–j edges*, that is, edges from area i to area j, by $n_{ij}(S, T)$ (for $i, j \in \{0, 1, 2, 3\}$). Figure 7.9a shows the four groups of i–j edges for $i = 1$.

The only crossings that our algorithm does not take into account are crossings between edges whose lowest common ancestors lie in parts of S_0 and T_0 that are split apart into different branches of the recursion. For the subinstance $\langle S, T\rangle$, which is split into $\langle S_1, T_1\rangle$ and $\langle S_2, T_2\rangle$, this means that for all $n_{12}(S, T)$ edges that run between S_1 and T_2, we fail to consider all crossings between pairs of two such edges. Similarly, we do not consider any pair of the $n_{21}(S, T)$ edges between S_2 and T_1.

Let's return to the drawing δ and consider the set $\mathcal{I}$ of subinstances that correspond to δ, that is, all pairs of opposing subtrees in δ. For each subinstance $\langle S, T\rangle \in \mathcal{I}$ we do not account for crossings of pairs of 1–2 edges and pairs of 2–1 edges since these edges run between two subinstances that are solved independently. In the worst case all these edge pairs cross and our algorithm misses $\binom{n_{12}(S,T)}{2} + \binom{n_{21}(S,T)}{2}$ crossings. Let c_δ be the number



(a) Edges incident to $L(S_1)$ are separated into four groups by the area of their second endpoint.

(b) All edge groups that cross the n_{12} 1–2 edges between $L(S_1)$ and $L(T_2)$.

Figure 7.9: Areas of the endpoints and types of edges incident to $L(S)$ and $L(T)$. Cardinalities $n_{ij}(S,T)$ are abbreviated as n_{ij}.

of crossings of δ counted by the algorithm, and let $|\delta|$ be the actual number of crossings of δ. Clearly, we have $c_\delta \leq |\delta|$. We can bound $|\delta|$ from above by

$$|\delta| \leq c_\delta + \sum_{\langle S,T\rangle \in \mathcal{I}} \left[\binom{n_{12}(S,T)}{2} + \binom{n_{21}(S,T)}{2} \right] \leq c_\delta + \sum_{\langle S,T\rangle \in \mathcal{I}} \frac{n_{12}^2(S,T) + n_{21}^2(S,T)}{2}. \quad (7.1)$$

We now show that $\sum_{\langle S,T\rangle \in \mathcal{I}} (n_{12}^2(S,T) + n_{21}^2(S,T)) \leq 2c_\delta$. For the sake of convenience we abbreviate $n_{ij}(S,T)$ as n_{ij} in the following. We will bound n_{12}^2 by the number of crossings of the 1–2 edges in δ that are counted by the algorithm. This number is at least

$$c_{12} = n_{12} \cdot (n_{03} + n_{20} + n_{21} + n_{30} + n_{31}) \quad (7.2)$$

as can be seen in Figure 7.9b. All these crossings are current-level crossings at this or some earlier point in the algorithm. Since our (sub)trees are complete and thus S_1 and T_1 have the same number of leaves, we obtain

$$n_{10} + n_{12} + n_{13} = n_{01} + n_{21} + n_{31}. \quad (7.3)$$

Furthermore, we have the following equality for the edges from areas 0 on both sides

$$n_{01} + n_{02} + n_{03} = n_{10} + n_{20} + n_{30}. \quad (7.4)$$

From (7.3) we obtain $n_{12} \leq n_{01} - n_{10} + n_{21} + n_{31}$ and from (7.4) we obtain $n_{01} - n_{10} \leq n_{20} + n_{30}$. Hence we have $n_{12} \leq n_{20} + n_{30} + n_{21} + n_{31}$. With (7.2) this yields

$$n_{12}^2 \leq n_{12} \cdot (n_{20} + n_{30} + n_{21} + n_{31}) \leq c_{12}, \quad (7.5)$$

that is, n_{12}^2 is bounded by the number of crossings that involve a 1–2 edge in δ and that are counted by the algorithm. Analogously, there are at least

$$c_{21} = n_{21} \cdot (n_{02} + n_{03} + n_{12} + n_{13} + n_{30}) \quad (7.6)$$

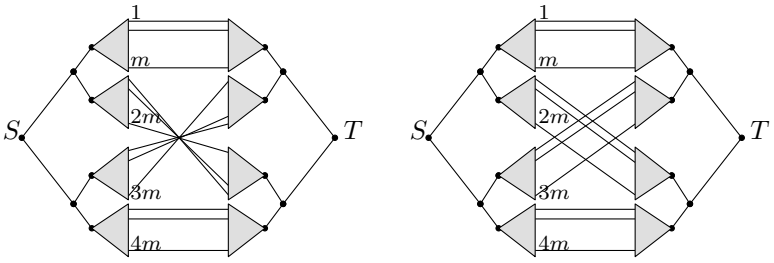


Figure 7.10: Example of a tanglegram for which our algorithm may output a drawing (left) that has roughly twice as many crossings as the optimal drawing (right).

counted crossings in δ that involve a 2–1 edge. From (7.3) we obtain $n_{21} \leq n_{10} - n_{01} + n_{12} + n_{13}$ and from (7.4) we obtain $n_{10} - n_{01} \leq n_{02} + n_{03}$. It follows that

$$n_{21}^2 \leq n_{21} \cdot (n_{02} + n_{03} + n_{12} + n_{13}) \leq c_{21}, \tag{7.7}$$

that is, n_{21}^2 is bounded by the number of crossings counted by the algorithm that involve a 2–1 edge in δ.

So from (7.5) and (7.7) we have $n_{12}^2 \leq c_{12}$ and $n_{21}^2 \leq c_{21}$. Applying this argument to all subinstances $\langle S, T\rangle \in \mathcal{I}$ we get

$$\sum_{\langle S,T\rangle\in\mathcal{I}} (n_{12}^2(S,T) + n_{21}^2(S,T)) \leq \sum_{\langle S,T\rangle\in\mathcal{I}} c_{12}(S,T) + \sum_{\langle S,T\rangle\in\mathcal{I}} c_{21}(S,T) \leq 2 \cdot c_\delta. \tag{7.8}$$

The fact that $\sum_{\langle S,T\rangle\in\mathcal{I}} c_{12}(S,T) \leq c_\delta$ holds because each edge crossing in δ appears in at most one term $c_{12}(S,T)$. Let ab be a 1–2 edge in the subinstance $\langle S, T\rangle$. Then in all parent instances of the recursion, ab was still a 1–1 edge or a 2–2 edge; such edges do not appear in any previous c_{12}-term. In a subsequent instance $\langle S', T'\rangle$ below $\langle S, T\rangle$ in the recursion the edge ab might in fact reappear, for example as a 0–3 edge. At that point, however, it is considered as an edge that crosses one of the 1–2 edges of $\langle S', T'\rangle$, say cd. But then cd was considered as a 1–1 or 2–2 edge in all previous instances. Hence the crossing between ab and cd does not appear in any other c_{12}-term. Analogous reasoning yields $\sum_{\langle S,T\rangle\in\mathcal{I}} c_{21}(S,T) \leq c_\delta$

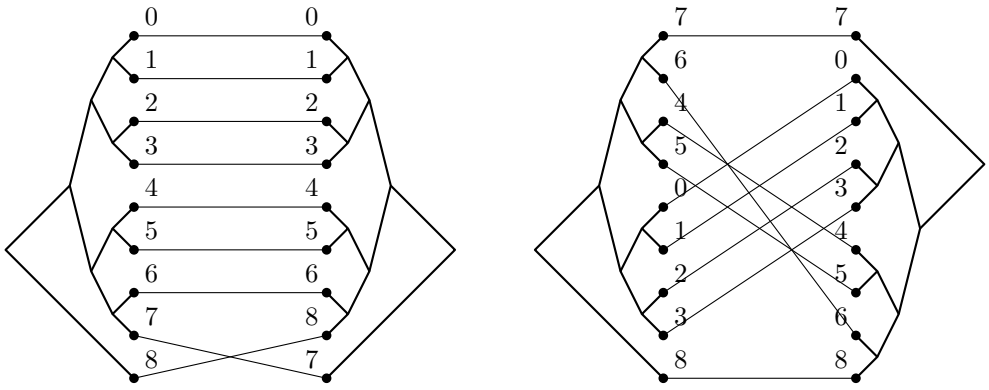
Plugging (7.8) into (7.1) yields $|\delta| \leq 2c_\delta$. Now let $A^\star$ be the solution computed by our algorithm and let $S^\star$ be an optimal solution. Their actual numbers of crossings are denoted as $|A^\star|$ and $|S^\star|$, respectively. By $c_{A^\star}$ and $c_{S^\star}$ we denote the number of crossings counted by our algorithm for the drawings $A^\star$ and $S^\star$, respectively. Since $|\delta| \leq 2c_\delta$ for any drawing δ we get

$$|A^\star| \leq 2c_{A^\star} \leq 2c_{S^\star} \leq 2|S^\star|,$$

that is, the algorithm is indeed a factor-2 approximation. □

We note that the approximation factor of 2 is tight for our algorithm: let $n = 4m$, let S have leaves ordered $1, \ldots, 4m$, and let T have leaves ordered $1, \ldots, m, 3m, \ldots, 2m+1$, $m+1, \ldots, 2m, 3m+1, \ldots, 4m$ (see Figure 7.10). Then our algorithm may construct a drawing with $m^2 + 2\binom{m}{2} = 2m^2 - m$ crossings, while the optimal drawing has only m^2 crossings.

Generalization to *d*-ary trees. The algorithm can be generalized to complete d-ary trees. The recurrence relation of the running time changes to $T(n) \leq d \cdot (d!)^2 \cdot T(n/d) + O(n)$ since we need to consider all $d!$ subtree orderings of both trees, each triggering d subinstances



(a) Optimal layout: 1 crossing. (b) Heuristic layout: 14 crossings.

Figure 7.11: Example for which Algorithm 7.1 performs badly as a heuristic.

of size n/d. This resolves to $T(n) = O(n^{1+2\log_d(d!)})$. For $d \geq 3$ the running time is upper-bounded by $O(n^{2d-1.7})$. At the same time the approximation factor increases to $1 + \binom{d}{2}$. This is because for any pair (i, j) with $1 \leq i < j \leq d$ the algorithm fails to account for potential crossings between the trees S_i and T_j as well as between S_j and T_i. This number can be bounded for each of the $\binom{d}{2}$ pairs by the number of crossings in the optimal solution using our arguments for binary trees.

Heuristic improvements for general binary trees. In applications most binary TL instances do not consist of *complete* binary trees. The above recursive algorithm can be applied to any pair of binary trees S and T as a heuristic but an approximation guarantee cannot be given any more. Moreover, the running time grows exponentially with the height of the trees. We denote the smaller of the two heights of S and T by h. For example, two caterpillar trees with n leaves yield a running time of $O(n \cdot 2^n)$. Still, there is room for practical improvements of running time and solution quality for arbitrary binary tanglegrams.

The original algorithm always divides an instance into an upper and a lower subinstance, see Figure 7.8. For unbalanced trees this may lead to an unnecessarily high number of ignored crossings as the example in Figure 7.11 shows. Algorithm 7.1 selects the subtree order at both roots that aligns leaves 7 and 8, respectively, as shown in Figure 7.11b. This choice does not cause any current-level crossings and since both subinstances contain a leaf the algorithm stops. Even in the best case this decision causes 14 crossings. In contrast, the optimal layout contains only one crossing, see Figure 7.11a.

A small modification of the algorithm weakens this effect (and yields the optimal solution in the given example). Instead of defining a subinstance by a pair of subtrees on the same horizontal level we match the four subtrees S_1, S_2, T_1, and T_2 of v_S and v_T such that the number of inter-tree edge pairs within each of the two subinstances is larger than the number of edge pairs going from one subinstance to the other. In this way less edge pairs that potentially cross are disregarded in the course of the algorithm. This not only improves the performance but it also means that each subtree of S has a fixed partner in T for all branches of the recursion. We precompute in $O(n^2)$ time all required current-level crossings for $O(h)$-time lookup according to Lemma 7.1. The general recurrence for two trees with n and m leaves is $T(n, m) \leq 4T(n-k_1, m-k_2)+4T(k_1, k_2)+O(h)$, where $k_1, k_2 \geq 1$ denote the sizes of the two subtrees in one of the subinstances. The base case is $T(n, 1) = T(1, n) = O(nh)$.

On the one hand, for complete binary trees we have $n = m$ and $k_1 = k_2 = n/2$, which resolves to $T(n, n) \in O(n^3)$ as for the unmodified algorithm. Note that the approximation factor of 2 is no longer guaranteed for the modified algorithm; however, our experiments with complete binary trees showed that on average the modified algorithm is even slightly better than the unmodified one. On the other hand, for the worst-case instance of two caterpillar trees the recurrence becomes $T(n, n) \leq 4T(n-1, n-1) + O(n)$, which resolves to $T(n) \in O(4^n)$. The running time for arbitrary binary trees depends on their balancedness and ranges between $O(n^3)$ and $O(4^n)$.

In order to speed up the algorithm in practice we additionally make use of a branch-and-bound technique in order to prune large parts of the search tree as early as possible. For a subinstance $\langle S, T \rangle$ with roots v_S and v_T we first consider the arrangement of the subtrees of v_S and v_T that yields the lowest number of accumulated current-level crossings and recurse on its two subinstances. Once the leaf level is reached this gives us an initial upper bound on the number of crossings. Now at each node of the search tree we can immediately prune all subtrees corresponding to arrangements of the current subinstance whose accumulated current-level crossings exceed this upper bound. The rest of the search tree is examined further, and each time a better solution is found, the upper bound is updated accordingly.

Maximization version.

Instead of the original TL problem, which minimizes the number of pairs of edges that cross each other, we may consider the dual problem TL* of maximizing the number of pairs of edges that do not cross. The tasks of finding optimal solutions for these problems are equivalent, but from the perspective of approximation it makes quite a difference which of the two problems we consider. Here we do not assume that we draw *binary* trees. Instead, if an inner node has more than two children, we assume that we may choose only between a given permutation of the children and the reverse permutation obtained by flipping the whole block of children.

In contrast to the TL problem, which is hard to approximate as we have shown in Theorem 7.1, the TL* problem has a constant-factor approximation algorithm. We show this by reducing TL* to a constrained version of the MaxCut problem, which can be approximately solved with the semidefinite programming rounding algorithm of Goemans and Williamson [GW95].

Theorem 7.4 *There exists a* 0.878*-approximation for the TL* problem.*

Proof. Fix any input drawing of the two trees S and T in an instance of the TL* problem. As before we can associate a decision variable to any inner node of each of the trees. The variable decides whether we flip the block of children at the corresponding node or not. We model this situation by a weighted graph $G = (V, E)$; a flip decision corresponds to deciding to which side of a cut the corresponding vertex is assigned.

For each inner node v of a tree in the instance $\langle S, T \rangle$ of TL* the constructed graph G contains two vertices v and v'. For each pair ab and cd of inter-tree edges with $a, c \in L(S)$ and $c, d \in L(T)$, we have a weighted edge in E, as follows. Let $v = \mathrm{lca}(a, c)$ and $w = \mathrm{lca}(b, d)$ be the lowest common ancestors of the edge pair. If ab and cd cross in the initial drawing, then we add the edge vw with weight 1 in G (or increase its weight by one if it is already present). If they do not cross in the initial drawing, then we analogously add the edge vw' in G or increase its weight by one.

It remains to observe that cuts in G that separate each pair (v, v') correspond to drawings of S and T in the instance of the TL* problem. To see this choose one half C of such a cut $(C, V \setminus C)$. For a vertex $v \in C$ we do not flip the children of the corresponding node in $\langle S, T\rangle$ and for a vertex $v' \in C$ we do flip the children. (The other half of the cut yields the mirrored drawing with the same number of crossings.) Moreover, edges that are cut in $(C, V \setminus C)$ correspond to pairs of edges that do not cross in the drawing of the two trees.

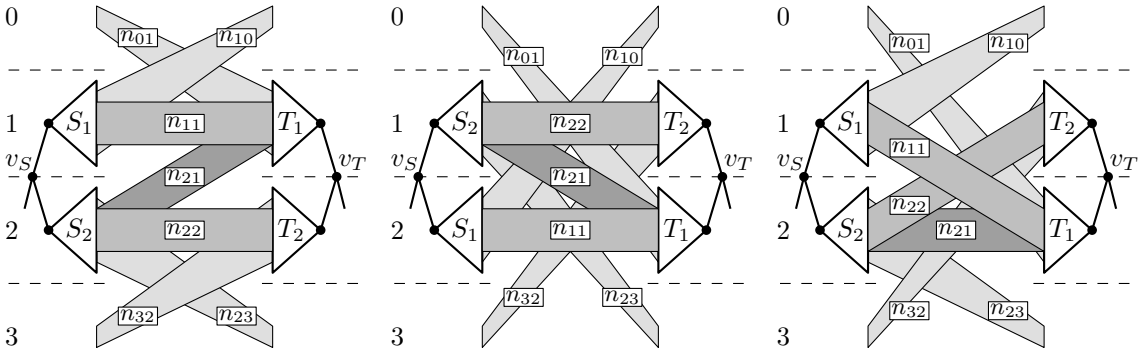
The resulting optimization problem is the MaxResCut problem (that is, MaxCut with additional constraints forcing certain pairs of vertices to be separated by the cut) studied by Goemans and Williamson [GW95]. Therefore, we may use their semidefinite programming rounding algorithm to compute a 0.878-approximation of the largest constrained cut in the graph G. This cut determines which of the subtrees in the initial drawing must be flipped to obtain a drawing that is a 0.878-approximation to TL*. □

7.4.2 Fixed-Parameter Algorithm

We consider the following parameterized variant of the TL problem. Given a complete binary TL instance $\langle S, T\rangle$ and a non-negative integer k, decide whether there exists a layout of S and T with at most k induced inter-tree edge crossings. Our algorithm makes use of the same technique to count current-level crossings as the 2-approximation algorithm. Hence we precompute the crossing tables $C^=$ and $C^\times$ in $O(n^2)$ time as before, see Lemma 7.1. The algorithm traverses the inner nodes of S in breadth-first order. It starts at the root of S and its corresponding node in T (in this case the root of T), branches into all four possible subtree configurations (at the root it actually suffices to consider two of them), and subtracts from k the number of incurred crossings in each branch. Then we proceed recursively with the next node v in S, its corresponding opposite node w in T, and the reduced parameter k' of allowed crossings. In each node of the search tree we count the current-level crossings for each of the subtree orders of v and w by summing up in linear time the appropriate entries in $C^=$ and $C^\times$ for v (or w) and all of the $O(n)$ subtree orders that are already fixed in T (or S). Once we reach a leaf of the search tree we know the exact number of crossings since each pair of edges ab and cd is counted as soon as the subtree orders of both $\mathrm{lca}(a, c)$ and $\mathrm{lca}(b, d)$ are fixed. Obviously, we stop following a branch of the search tree when the parameter value drops below 0.

For the search tree to have bounded height, we need to ensure that whenever we move to the next subinstance, the parameter value decreases at least by one. At first sight this seems problematic: if a subinstance does not incur any current-level crossings, the parameter will not drop. The following key lemma—which does not hold for general binary trees—shows that there is a way out. It says that if there is an order of the subtrees in a subinstance that does not incur any current-level crossings, then we can ignore the other three subtree orders and do not have to branch.

Lemma 7.2 *Let $\langle S, T\rangle$ be a complete binary TL instance, and let v_S be a node of S and v_T a node of T such that v_S and v_T have the same distance to their respective root. Further, let (S_1, S_2) be the subtrees incident to v_S and let (T_1, T_2) be the subtrees incident to v_T. If the subinstance $\langle (S_1, S_2), (T_1, T_2)\rangle$ does not incur any current-level crossings, then each of the subinstances $\langle (S_1, S_2), (T_2, T_1)\rangle$, $\langle (S_2, S_1), (T_1, T_2)\rangle$, and $\langle (S_2, S_1), (T_2, T_1)\rangle$ has at least as many crossings as $\langle (S_1, S_2), (T_1, T_2)\rangle$, for any fixed ordering of the leaves of S_1, S_2, T_1 and T_2.*



(a) $\langle (S_1, S_2), (T_1, T_2) \rangle$ (b) $\langle (S_2, S_1), (T_2, T_1) \rangle$ (c) $\langle (S_1, S_2), (T_2, T_1) \rangle$

Figure 7.12: Edge types and crossings of the instance $\langle S, T \rangle$. Only non-empty classes of edge types are shown.

Proof. If the subinstance $\langle (S_1, S_2), (T_1, T_2) \rangle$ does not incur any current-level crossings, this excludes certain types of edges. We categorize the inter-tree edges originating from the four subtrees according to their destinations as before and use the notation n_{ij} for the number of edges between area i on the left and area j on the right—see Figure 7.12a. First of all, there are no edges between S_1 and T_2 or between S_2 and T_1. We consider only the first case, where $n_{12} = 0$; the second case $n_{21} = 0$ is symmetric. In both cases, we have $n_{13} = n_{31} = n_{20} = n_{02} = 0$. Since we consider complete binary trees, we obtain $n_{10} = n_{01} + n_{21}$, $n_{32} = n_{23} + n_{21}$, and $n_{01} + n_{11} = n_{23} + n_{22}$.

We fix an ordering σ of the leaves of the four subtrees S_1, S_2, T_1, and T_2. We first compare the number of crossings in the subinstance $\langle (S_1, S_2), (T_1, T_2) \rangle$ with the number of crossings in the subinstance $\langle (S_2, S_1), (T_2, T_1) \rangle$, see Figures 7.12a and 7.12b. The subinstance $\langle (S_1, S_2), (T_1, T_2) \rangle$ can have at most $n_{21}(n_{11} + n_{22})$ crossings that do not occur in $\langle (S_2, S_1), (T_2, T_1) \rangle$. However, $\langle (S_2, S_1), (T_2, T_1) \rangle$ has at least $n_{10}(n_{23} + n_{21} + n_{22}) + n_{23}n_{11} + n_{32}(n_{01} + n_{21} + n_{11}) + n_{01}n_{22}$ crossings that do not appear in $\langle (S_1, S_2), (T_1, T_2) \rangle$. Plugging in the above equalities for n_{10} and n_{32}, we get $(n_{01} + n_{21})(n_{23} + n_{21} + n_{22}) + n_{23}n_{11} + (n_{23} + n_{21})(n_{01} + n_{21} + n_{11}) + n_{01}n_{22} \geq n_{21}(n_{11} + n_{22})$. Thus, the subinstance $\langle (S_2, S_1), (T_2, T_1) \rangle$ has at least as many crossings with respect to the fixed leaf order σ as $\langle (S_1, S_2), (T_1, T_2) \rangle$ has.

Next, we compare the number of crossings in the subinstance $\langle (S_1, S_2), (T_1, T_2) \rangle$ with the number of crossings in the subinstance $\langle (S_1, S_2), (T_2, T_1) \rangle$, see Figures 7.12a and 7.12c. Now the number of additional crossings of $\langle (S_1, S_2), (T_1, T_2) \rangle$ is at most $n_{21}n_{22}$, and the subinstance $\langle (S_1, S_2), (T_2, T_1) \rangle$ introduces at least $(n_{01} + n_{11})(n_{32} + n_{22}) + n_{32}n_{21}$ additional crossings. With the equality $n_{01} + n_{11} = n_{23} + n_{22}$ and the inequality $n_{32} + n_{22} \geq n_{21}$ we get $(n_{01} + n_{11})(n_{32} + n_{22}) + n_{32}n_{21} \geq (n_{23} + n_{22} + n_{32})n_{21} \geq n_{22}n_{21}$. Thus, the subinstance $\langle (S_1, S_2), (T_2, T_1) \rangle$ has at least as many crossings with respect to σ as $\langle (S_1, S_2), (T_1, T_2) \rangle$ has.

By symmetry, the same holds for the last case $\langle (S_2, S_1), (T_1, T_2) \rangle$, which incurs at least as many crossings as $n_{11}n_{21}$, the number of crossings that can be present in $\langle (S_1, S_2), (T_1, T_2) \rangle$ but not in $\langle (S_2, S_1), (T_1, T_2) \rangle$. □

Counting the current-level crossings takes $O(n)$ time for each node that fixes its subtree order. If an order does not incur any current-level crossings we might need to fix in total up to $O(n)$ subtree orders and count the incurred crossings until we reach a new node of the

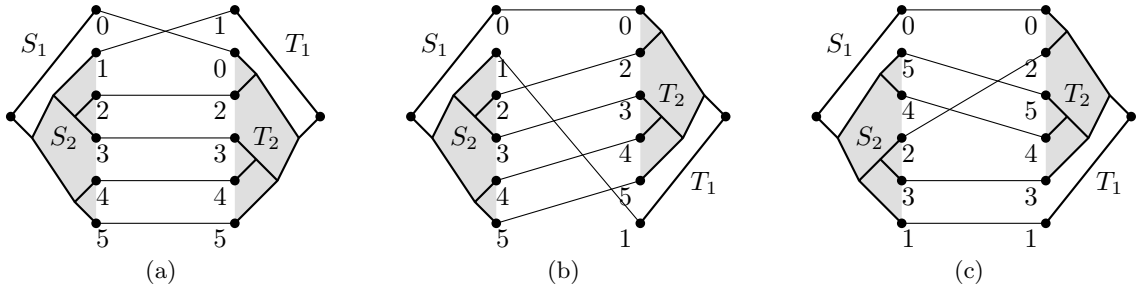


Figure 7.13: Example of a TL instance with an optimal layout that has one crossing (a). The same order of the leaves in the subtrees S_2 and T_2 yields four crossings for a configuration without current-level crossings (b). The best layout that avoids the current-level crossing still has two crossings (c).

search tree. Thus we spend $O(n^2)$ time for each of the $O(4^k)$ search-tree nodes. Including the preprocessing this yields a total running time of $O(n^2 + 4^k n^2)$. If the algorithm reaches a leaf of the search tree it has fixed all subtree orders in S and T and thus found a layout of the input instance that has at most k inter-tree edge crossings. If the search stops without reaching a leaf there is no layout of $\langle S, T\rangle$ with at most k inter-tree edge crossings.

Theorem 7.5 *Given a complete binary TL instance $\langle S, T\rangle$ with n leaves in each tree and an integer k we can in $O(4^k n^2)$ time determine either a layout of $\langle S, T\rangle$ with at most k inter-tree edge crossings or report that no such layout exists.*

Finally, the fact that Lemma 7.2 relies on the completeness of the two trees is illustrated in Figure 7.13. Here we have an example of an instance whose optimal layout requires a current-level crossing (Figure 7.13a). At the same time, the configuration $\langle (S_1, S_2), (T_2, T_1)\rangle$ has no current-level crossing. According to Lemma 7.2 the leaf order of the optimal layout copied into the layout without current-level crossings would produce at most as many crossings as in the other layout. Figure 7.13b shows that this is not true in our example. The best solution of the configuration $\langle (S_1, S_2), (T_2, T_1)\rangle$ still has two crossings and is not optimal (Figure 7.13c). Hence, we do have to consider *all* subtree orders even if one of them incurs no current-level crossings. This means that we cannot bound the size of the search tree in terms of the parameter k as we have done for complete binary trees.

7.4.3 Exact Algorithms

In this section we give two methods to compute optimal solutions for general binary TL exactly. The first is a simple formulation of the problem as an integer linear program (ILP). The second method is a branch-and-bound algorithm.

Integer Linear Program

Let $\langle S, T\rangle$ be an instance of general binary TL with an arbitrary input drawing. For each inner node $u \in S^\circ \cup T^\circ$ we introduce a binary variable x_u. If $x_u = 1$, the two subtrees of u change their order with respect to the input drawing, otherwise the order of the input drawing is kept. Any assignment of these variables corresponds to a tanglegram layout. Let ab and cd be two inter-tree edges with $a, c \in S$ and $b, d \in T$ and let $v = \mathrm{lca}(a, c)$ and $w = \mathrm{lca}(b, d)$ be their lowest common ancestors. Following Observation 7.1 we

distinguish two cases: If ab and cd cross in the input layout, they cross in any layout that either swaps the subtrees of both v and w or that does not swap either of them. In other words, there is a crossing if and only if $x_v = x_w$. Similarly, if ab and cd do not cross in the input layout, they cross in any layout that swaps the subtrees of exactly one of v or w, or, equivalently, ab and cd cross if and only if $x_v \neq x_w$.

Now for a pair $(v, w) \in S^\circ \times T^\circ$, let $k^\times_{vw}$ ($k^=_{vw}$) be the number of edge pairs that have v and w as their lowest common ancestors and that do (do not) cross in the initial layout. Note that $k^\times_{vw}$ and $k^=_{vw}$ are constant and can be computed for all pairs (v, w) in $O(n^2)$ time (see the computation of the tables $C^=$ and $C^\times$ in Lemma 7.1). Finally, we would like to define a variable y_{vw} for each pair $(v, w) \in S^\circ \times T^\circ$ that equals 1 if $x_v \neq x_w$ and 0 otherwise. This can be achieved with four linear constraints and yields the following ILP formulation for binary TL:

$$\text{Minimize} \quad \sum_{v \in S^\circ,\, w \in T^\circ} [y_{vw} \cdot k^=_{vw} \;+\; (1 - y_{vw}) \cdot k^\times_{vw}]$$

$$\begin{aligned} \text{subject to} \quad & y_{vw} \leq 2 - x_v - x_w \\ & y_{vw} \leq x_v + x_w \\ & y_{vw} \geq x_v - x_w \\ & y_{vw} \geq x_w - x_v \end{aligned} \qquad \text{for all } v \in S^\circ,\, w \in T^\circ.$$

Branch-and-Bound Algorithm

The branch-and-bound idea used in the implementation of the recursive splitting heuristic in Section 7.4.1 can also be applied to compute optimal solutions. The main ingredients for speeding up the exhaustive search are: (i) ordering the nodes in the search tree according to the impact of the swapping decisions to quickly find good solutions and (ii) using lower bounds that are as tight as possible to cut off large parts of the search tree as early as possible.

We preprocess the given binary TL instance $\langle S, T \rangle$ by computing for any pair $(v, w) \in S^\circ \times T^\circ$, the crossing table entries $C^\times[v, w]$ and $C^=[v, w]$ in $O(n^2)$ time (see Lemma 7.1). We further store an *interaction counter* $\mathrm{ic}(v)$ for each node $v \in S^\circ \cup T^\circ$ that contains the number of inner nodes w in the opposite tree for which $|C^\times[v, w] - C^=[v, w]| > 0$ (otherwise the swapping decisions for v and w are independent). Then we construct and traverse the search tree starting at the node whose interaction counter has the largest value. At each subsequent step we consider the next unvisited node $v \in S^\circ \cup T^\circ$ that has the largest difference between the number of crossings induced by swapping and by not swapping its subtrees given all the previous decisions in the search tree. If $\mathrm{ic}(v) = 0$, we simply assign the better choice for v since no further decisions depend on v. Otherwise we compute for both swap options at v a lower bound on the number of crossings arising from all subsequent nodes of $S \cup T$. This is done by summing up in linear time the respective minima of the two possible crossing numbers at each node given the decisions taken so far in the traversal of the search tree, including v. If the sum of this lower bound on future crossings and the number of current crossings is greater or equal to the current best solution, we can safely prune the current branch of the search tree. Otherwise we update the number of induced crossings for the remaining nodes accordingly (using the precomputed crossing numbers), decrease the interaction counters for the remaining nodes that interact with v by one, and go further down the search tree. Once all nodes of the search tree have been visited or cut off, we return the best solution found.

The running time of the algorithm is $O(n^2+n\cdot 2^{2n})$ in the worst case since we spend $O(n)$ time per node of the search tree.

7.4.4 Greedy Heuristic

The order, in which the nodes are visited in the exact branch-and-bound algorithm also gives rise to a simple greedy heuristic. We start to fix the subtree order of the node v in $S^\circ \cup T^\circ$ that has the largest interaction counter and update the induced crossing numbers for both subtree orders of the remaining nodes in $S^\circ \cup T^\circ$. The next node to fix its subtree order is the node v' that maximizes the difference between the number of crossings induced by swapping and not swapping its subtrees. We choose the order that yields the smaller number of crossings. After updating the crossing numbers of the remaining nodes given the subtree order at v' we proceed with the next unvisited node that maximizes the crossing difference until we have fixed all subtree orders. Note that this layout is exactly the same as the first solution obtained by the exact branch-and-bound algorithm.

The greedy algorithm requires $O(n^2)$ time for the preprocessing phase. For each of the $O(n)$ steps of the algorithm itself we need $O(n)$ time to find the next node in the order and to update all induced crossing numbers. Hence the greedy algorithm runs in $O(n^2)$ time.

7.5 Experimental Evaluation

We have implemented those algorithms of Section 7.4 that can be applied to general binary tanglegrams in Java 1.6. We have executed them on an AMD Opteron 2218 2.6 GHz system with 8 GB RAM under SuSE Linux 10.3. For solving the ILP we used the Java API of the commercial mathematical programming software Ilog CPLEX 11.1[1]. The primary goal of our study is to evaluate which of the proposed algorithms best solves binary TL for real-world instances. The most important criterion is thus the performance ratio with respect to the optimal solution in terms of the number of crossings. A secondary goal is to identify algorithms that are fast enough to be used interactively in a tanglegram visualization tool.

In the following we list the six algorithms taking part in our evaluation and describe the test data. Then we evaluate performance and running times of the different algorithms.

7.5.1 Algorithms in the Evaluation

The following six algorithms are included in our evaluation:

1. the recursive splitting algorithm in its heuristic variant *rec-split++* (Section 7.4.1)
2. the *hierarchy-sort* heuristic of Holten and van Wijk [HvW08] and its weighted variant *hierarchy-sort++* (described below)
3. the iterated one-tree crossing minimization algorithm *1tcm-iterated* of Dwyer and Schreiber [DS04] (described below)
4. the greedy heuristic *greedy* (Section 7.4.4)

[1] see `http://www.ilog.com/products/cplex`

5. the integer linear program *ilp* (Section 7.4.3)

6. the exact branch-and-bound algorithm *bb-exact* (Section 7.4.3)

We now sketch the two algorithms *hierarchy-sort* and *1tcm-iterated* that have been proposed previously in the literature before turning to the actual evaluation.

Hierarchy Sort

The hierarchy sort algorithm of Holten and van Wijk [HvW08] performs a number of collapse-and-expand cycles on both trees of the binary tanglegram $\langle S, T \rangle$. During each step of these cycles, the well-known barycentric method of Sugiyama et al. [STT81] for one-sided crossing minimization is used by successively fixing one tree, optimizing the leaf order of the other, and then changing the trees' roles until no further crossing reduction is possible.

In a first step the two trees are augmented with dummy nodes of degree 2 below the original leaves such that both trees have the same height and all leaves are on the lowest level. Starting at the lowest level, the barycenter method is applied to both trees in turn until the leaf order gets stable. Since the leaf order is restricted by the tree topology we need to consider only sibling nodes whose parent lies one level above. For such pairs of nodes we compute the barycenter of their neighbors in the other tree and swap them if the order of the two barycenters is reversed. In the next step we collapse the lowest level of both trees replacing each inter-tree edge between two leaves by the corresponding edge between their parents. Thus for higher levels multiple inter-tree edges can be incident to a single node. Barycentric crossing reduction and collapsing are alternated until the root of both trees is reached. Now the collapsing phase is replaced by an expansion phase that adds the next level to both trees. The initial leaf order of newly expanded subtrees remains as in the corresponding previous collapsing phase and then barycentric crossing reduction is performed on that level before expanding the next level. Once the lowest level is reached a collapse-and-expand cycle is completed. The collapse-and-expand cycles are repeated until no further improvement is made. Figure 7.14 is a step-by-step illustration of the algorithm.

We denote the algorithm of Holten and van Wijk [HvW08] as *hierarchy-sort*. A simple variant of the above procedure is to introduce integer weights on the edges during collapsing such that the edge weight corresponds to the number of original edges that are represented; we denote this variant as *hierarchy-sort++*. A further obvious variant is to reduce crossings at each level based on which configuration actually minimizes the resulting number of crossings rather than using the barycenter method. It turned out, however, that this variant performs worse than the weighted and unweighted barycenter heuristics.

The asymptotic running time of this algorithm depends of course on the number N of collapse-and-expand cycles and the maximum number N' of executions of the linear-time barycentric heuristic on each level. In our experiments it turned out that in all instances we had $N \leq 2$ and $N' \leq 4$ for the original heuristic. The weighted variant, on the other hand, occasionally got caught in an infinite loop of crossing reductions in one level of the algorithm, which needed to be aborted. Under the condition that both N and N' are constants, hierarchy sort runs in $O(n \cdot H)$ time, where H is the maximum height of the two trees. In the case of complete trees $H = \log n$, and the running time is $O(n \log n)$.

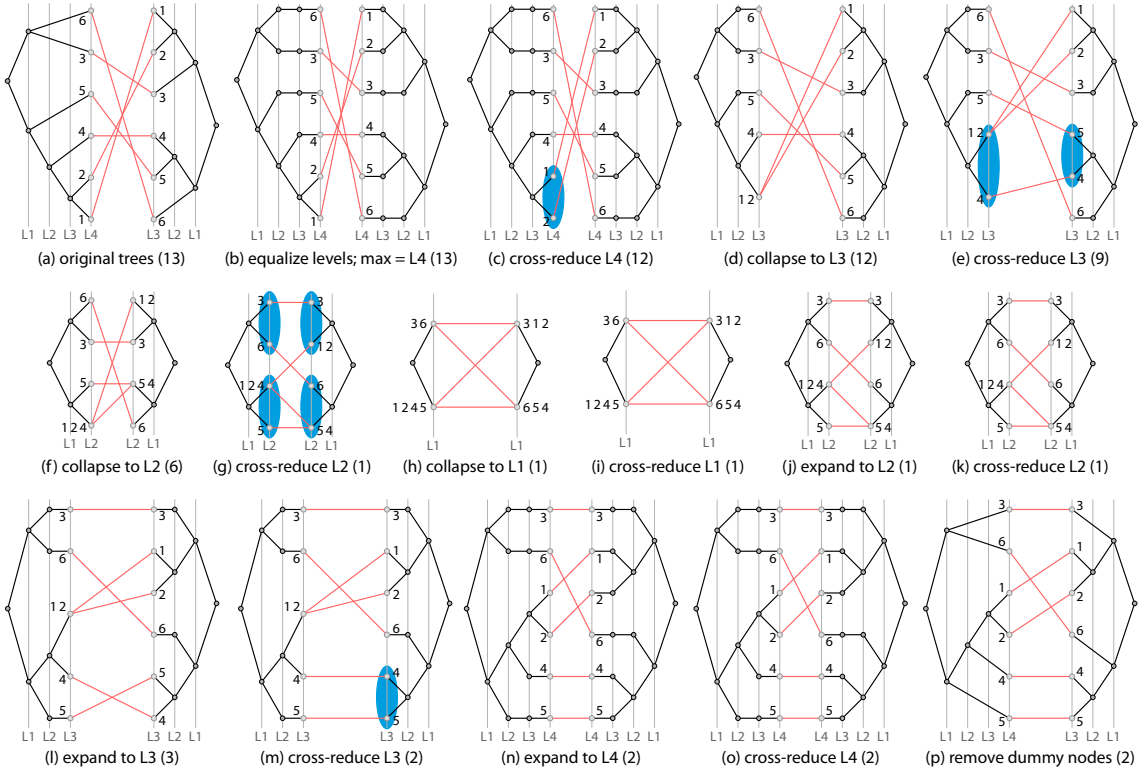


Figure 7.14: Step-by-step crossing reduction using *hierarchy-sort*. Nodes that are swapped during crossing reduction are encircled. The numbers of inter-tree edge crossings after each step are given in parentheses.

Iterated One-Tree Crossing Minimization

One-tree crossing minimization (1TCM) is the one-sided version of TL; one tree of a general binary tanglegram is fixed and the other is laid out optimally. An algorithm for this problem that runs in $O(n^2 \log n)$ time has first been described by Dwyer and Schreiber [DS04] and subsequently improved to $O(n \log^2 n)$ running time by Fernau et al. [FKP05]. We denote the tree to be optimized by T. The main observation is that the crossing behavior of any two inter-tree edges depends only on the swapping decision taken at their lowest common ancestor node v in T. Hence we can apply a divide-and-conquer strategy to recursively determine the optimal layout of the two subtrees of v and then arrange them at v such that the number of crossings caused by v is minimum. In our current implementation the divide-and-conquer algorithm still has a running time of $O(n^3)$. In order to use this method for the general two-sided problem Dwyer and Schreiber [DS04] suggested to apply the 1TCM-algorithm in turn to the two given trees until a local optimum for TL is found. This iterated heuristic is denoted as *1tcm-iterated*.

7.5.2 Data

We generated four sets (A–D) of random tanglegrams. Set A contains 100 pairs of complete binary n-leaf trees with random leaf orders for each $n \in \{16, 32, 64, 128, 256, 512\}$. In set B we simulate tree mutations by starting with two identical complete binary trees and then randomly swapping the positions of up to 20% of the leaves of one tree. This is done as follows: we first pick a leaf uniformly at random and then iteratively climb up the tree

with probability 0.75 in each step. From the node thus reached we climb back down and choose its left or right child with equal probability until we reach another leaf. This leaf and the leaf picked in the beginning are swapped. Thus the probability of two leaves being swapped decreases with their distance in the tree. Set C contains 100 pairs of general binary n-leaf trees for each $n \in \{20, 30, \ldots, 300\}$. The trees are constructed from a set of nodes, initially containing the n leaves, by iteratively joining two random nodes in a new parent node that replaces its children in the set. This process generates trees that resemble phylogenetic trees or clustering dendrograms. It mimics the hierarchical construction of these trees, which iteratively joins the two closest clusters or set of species in terms of an appropriate distance measure. In set D the first tree is constructed as in set C while the second tree in each tanglegram is a mutation of the first one, where up to 10% of the leaves can swap positions as done in set B and up to 25% of the subtrees can reattach to another edge. The edge for attaching the subtree is selected in a random walk starting at the subtree's old position. The walk continues with probability 0.75 and picks the left or right edge with equal probability. Trees in this set are of interest since real-world tanglegrams often consist of two related and rather similar trees.

Our real-world examples comprise three sets (E–G) of 1303 pairs of phylogenetic trees of animal gene families obtained from the TreeFam database[2] [LCR+06, Li06]. The trees in set E were generated from the multiple sequence alignments provided by TreeFam using the phylogenetic tree construction software treebest[3] [Li06]. For each database entry the first tree was built using the maximum-likelihood algorithm PHYML [GG03] and the second tree using the distance-based method neighbor joining (nj) [SN87]. Both methods are widely used in bioinformatics and have turned out to generate trees that are closest to the manually curated trees in TreeFam [Li06]. Neighbor-joining depends on a distance measure that reflects the probabilities of mutations at the positions in the underlying DNA or protein sequences. Two commonly used distances are the synonymous distance (*ds*) and the non-synonymous distance (*dn*). Distance *dn* is more appropriate for modeling long-term evolution while *ds* covers more recent mutation events better. The nj-trees in set E are actually constructed by a tree-merge algorithm described by Li [Li06] and implemented in treebest that builds a consensus tree of the nj-trees for the distances *ds* and *dn*. It is an obvious question to compare the merged tree with its two source trees. Therefore, sets F and G contain the tanglegrams consisting of the merged nj-tree and its underlying *ds*- or *dn*-tree, respectively. All three data sets E–G share the fact that about 75% of the trees have less than 50 leaves and only 5% have more than 100 leaves.

The crossing numbers of our examples are depicted in Figure 7.15. They vary strongly: as to be expected, mutated trees (B and D) have far lower crossing numbers than random pairs of trees (A and C). The TreeFam tanglegrams in sets E and G are generally characterized by low crossing numbers—at least for $n < 200$. Only set F and the largest trees in set E have relatively high crossing numbers ranging between those in sets C and D.

7.5.3 Performance

To each tanglegram we applied the five heuristics, the ILP, and the exact branch-and-bound algorithm *bb-exact*. We recorded the number k_i of crossings in the output, i being one of the heuristics in {*rec-split++*, *hierarchy-sort*, *hierarchy-sort++*, *1tcm-iterated*, *greedy*}. We recorded only results that were obtained within at most one minute wall clock time.

[2] `http://www.treefam.org`
[3] `http://treesoft.sourceforge.net`

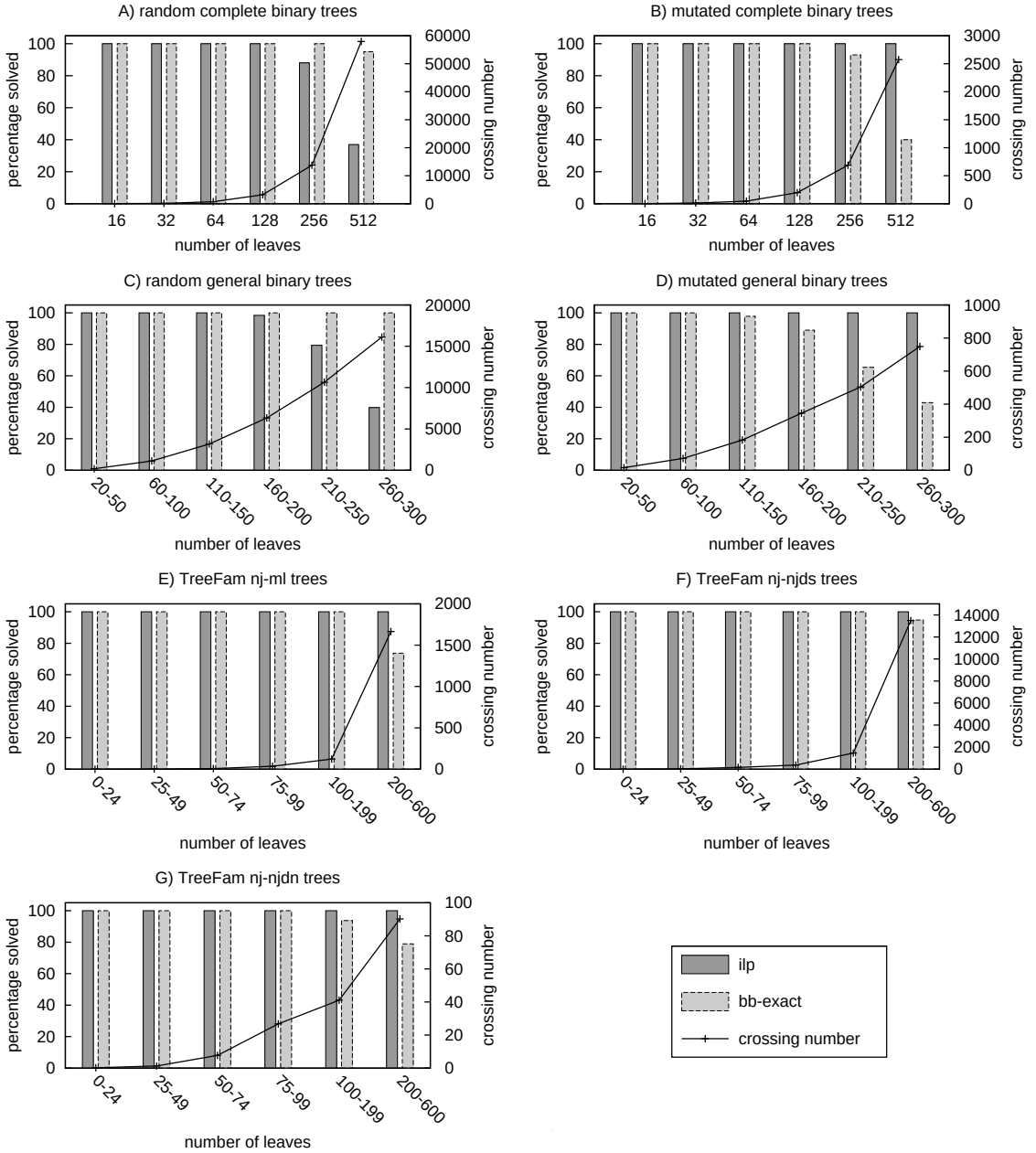


Figure 7.15: Percentage of solved instances for the exact algorithms *ilp* and *bb-exact* (left axes) and average crossing numbers (right axes) of random (A–D) and real-world (E–G) binary tanglegrams. Note the different scales on the right axes.

For each tanglegram we computed the performance ratio $(k_i + 1)/(k + 1)$. The crossing number k was obtained from one of the two exact algorithms. Note that we add one to both numerator and denominator such that the ratio is also defined for crossing-free instances.

Note that we have also implemented the original recursive splitting algorithm *rec-split* in addition to its variant *rec-split++*. Yet, we do not include it in our detailed evaluation for several reasons. First of all, *rec-split* is designed only for complete binary trees while *rec-split++* is adapted to deal with unbalanced trees as well. Hence *rec-split* performs badly on general binary tanglegrams, the focus of our evaluation. Moreover, for complete binary tanglegrams, both variants perform almost identically. Lastly, the running time of *rec-split* is far slower than the running time of the branch-and-bound variant *rec-split++* so that for $n \geq 512$ in the case of complete trees the instances could not be solved within the given time frame by *rec-split*. For general binary trees, this timeout was reached already for $n \geq 80$ in several instances.

In the subsequent discussion we refer to the performance ratios shown in Figure 7.16. A first inspection of the plots immediately reveals that there is a clear method of choice for all our examples that not only outperforms the other heuristics but even achieves average performance ratios hardly deviating from the optimum: *greedy*. This comes as a surprise since *greedy* is merely a byproduct of our exact branch-and-bound algorithm while the other heuristics were explicitly designed to obtain good solutions efficiently.

Next, we examine the results for the different sets of tanglegrams in more detail. We start with the complete binary trees in sets A and B. Set A with random pairs of trees shows that *hierarchy-sort* performs worst and spreads over a relatively large range of values. On the other hand the variant *hierarchy-sort++* is among the best heuristics apart from *greedy*. For $n \geq 64$ *rec-split++* and *1tcm-iterated* catch up with *greedy* and *hierarchy-sort++* and also achieve performance ratios between 1 and 1.1.

For set B containing mutated trees that are more similar to each other, the picture changes. Algorithms *greedy* and *rec-split++* outclass the the other three methods with average performance ratios below 1.01 and many optimal solutions while the other methods range between 1.6 and 4. In terms of outliers *rec-split++* is slightly preferable to *greedy*. Comparing sets A and B it is noteworthy that *rec-split++* and *greedy* perform equally well on random and mutated trees, while the remaining methods are susceptible to the similarity and consequently the crossing number of the two trees, see Figure 7.15. Recall that *rec-split++* is based on our 2-approximation algorithm *rec-split* for complete binary trees, but it is no 2-approximation itself; still, even the worst measured performance ratio of *rec-split++* in the experiments is far better than 2.

For the general binary trees in sets C and D the five heuristics have a similar ranking as for the sets A and B. Algorithm *greedy* remains the best method with average performance ratios below 1.01 and is even more clearly ahead of the remaining algorithms. For random pairs of trees (set C) the three methods *rec-split++*, *hierarchy-sort++*, and *1tcm-iterated* show a similar performance, which is on average below 1.1. The worst performance is again obtained by *hierarchy-sort*.

For mutated trees (set D) *greedy* is again almost optimal but this time *rec-split++* performs a lot better than the remaining competitors, which is similar to the situation for set B. While *rec-split++* shows performance ratios between 1 and 2 even for the third quartile, *hierarchy-sort++* and *1tcm-iterated* lie on average between 2 and 4. The original method *hierarchy-sort* even reaches average ratios close to 7. It should be noted that outliers for all algorithms except *greedy* reach values between 10 and 100. Furthermore, for random pairs of trees the completeness does not seem to affect the quality of the solutions

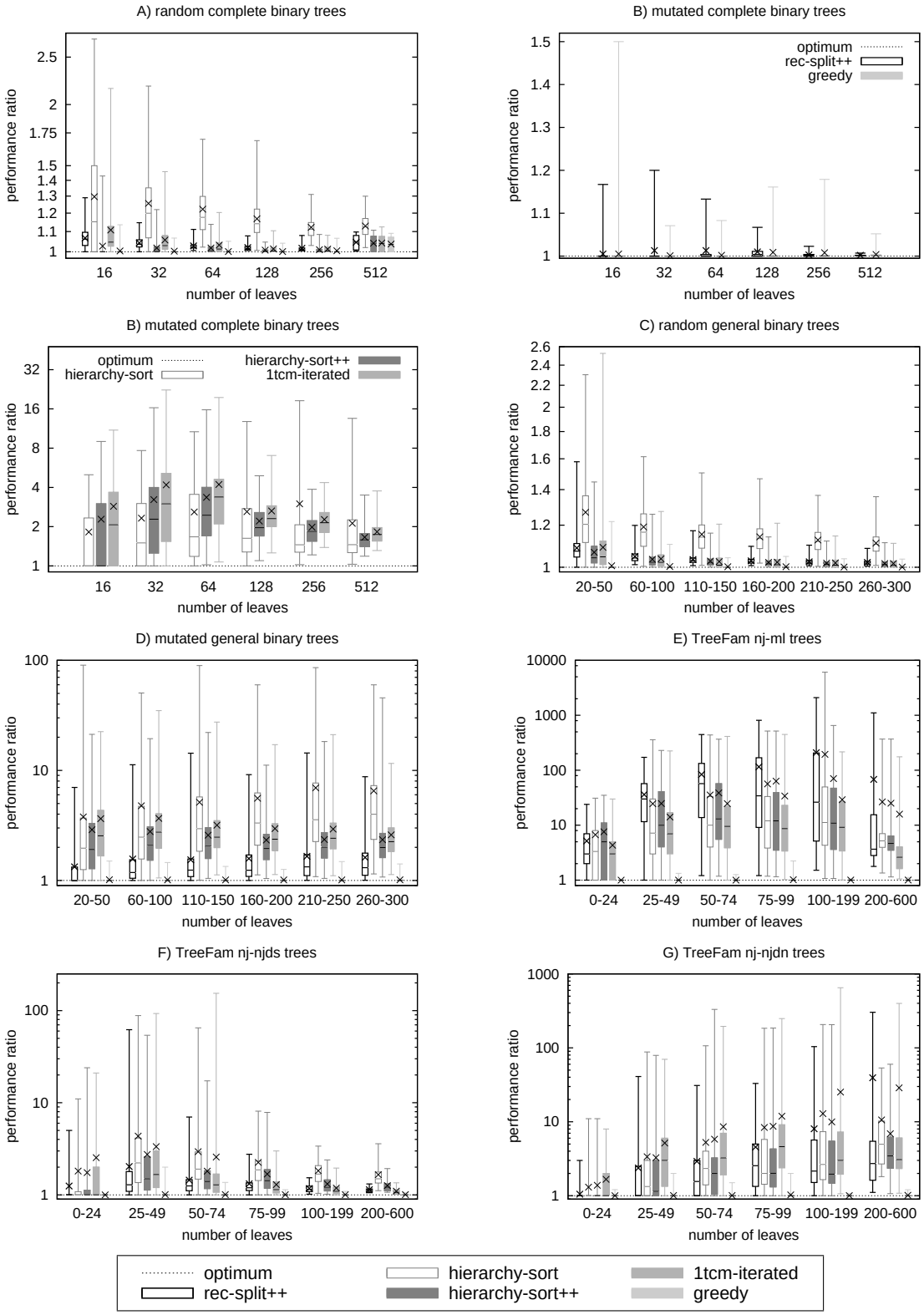


Figure 7.16: Performance ratios of the five algorithms *rec-split++*, *hierarchy-sort*, *hierarchy-sort++*, *1tcm-iterated*, and *greedy* for random (A–D) and real-world (E–G) binary tanglegrams. The boxplots show median (−), arithmetic mean (×), first and third quartile, minimum and maximum.

since the results for sets A and C are very similar. For mutated trees in set D, *rec-split++* becomes inferior to *greedy* unlike the results for set B. Also the performance ratios of the algorithms on the whole (except *greedy*) spread a lot more in set D than in set B.

The relative performance of the heuristics on the real-world data partially mirrors our observations from sets C and D but also exhibits different effects. In analyzing the results for sets E–G recall that about 95% of the trees have $n \leq 100$ leaves and that crossing numbers vary a lot, see Figure 7.15. Sets E and G contain pairs of rather similar trees. There, the crossing numbers for $n \leq 100$ are roughly the same, ranging between 0 and 35 on average. For trees of size $n > 100$, the crossing numbers in set E increase drastically; those in set G remain small. On the other hand, the crossing numbers in set F are higher by a factor of at least 10 in comparison to sets E and G.

The results for all three sets have in common that, as before, *greedy* attains by far the best performance ratios and even finds optimal solutions for over 75% of the instances. Moreover, the remaining heuristics have severe problems with outliers that reach inacceptable ratios between 100 and 1000, in some cases even worse. Note, however, that the ratio $(k_i{+}1)/(k{+}1)$ equals (almost) the absolute number of crossings k_i for instances with $k = 0$. In set E the second best method is *1tcm-iterated*, followed by the two hierarchy-sort variants. Interestingly, *hierarchy-sort++* is no longer preferable to *hierarchy-sort*. The algorithm *rec-split++* performs worst on set E. The order of the algorithms is thus quite different from that on the random set D although size and crossing number of the trees is similar.

Set F with less similar trees gives results that are comparable to set C in terms of the relative order of the algorithms. The *hierarchy-sort++* heuristic is worst, at least for $n \geq 25$. For instances with $n \leq 100$ *rec-split++* ranks second after *greedy*. Finally, *hierarchy-sort++* performs better than *1tcm-iterated* for small instances, while *1tcm-iterated* beats *hierarchy-sort++* on the large instances. In absolute numbers, however, outliers in set F are a lot worse than in set C.

Finally, set G shows results that are similar to set D, although the distinction between the heuristics (except *greedy*) is not as clear in set G. The main difference is that *hierarchy-sort* now keeps up with *hierarchy-sort++*, while it had been clearly worse in set D.

7.5.4 Running Time

Although the number of crossings is the main aspect for assessing the quality of TL algorithms, their running time is also an important criterion—especially if the layouts are to be produced interactively. Figure 7.17 shows plots of the median running times of our five heuristics as well as of the integer program *ilp* and the exact branch-and-bound algorithm *bb-exact*. In our experiments there was a timeout after one minute wall clock time for all algorithms. Note that the running times summarize regularly terminated runs and those aborted after one minute. Also note the different scales on the x-axes. The main question is whether *greedy*, whose performance ratio is far better than the other four heuristics and which even finds optimal solutions in most of the cases, is fast enough to be used in practice. Moreover, it is of interest whether we can afford to compute optimal solutions exactly for typical input sizes.

Let's first consider the running time of *greedy*. For small instances *greedy* is among the fastest methods with running times between 0.001 and 0.01 seconds. For larger instances the running times grow to values between 0.1 and 0.25 seconds, placing *greedy* in the mid-range of the other heuristics. Nonetheless, even the largest instances are solved in at most half a second. Further note that *greedy* has a worst-case running time of

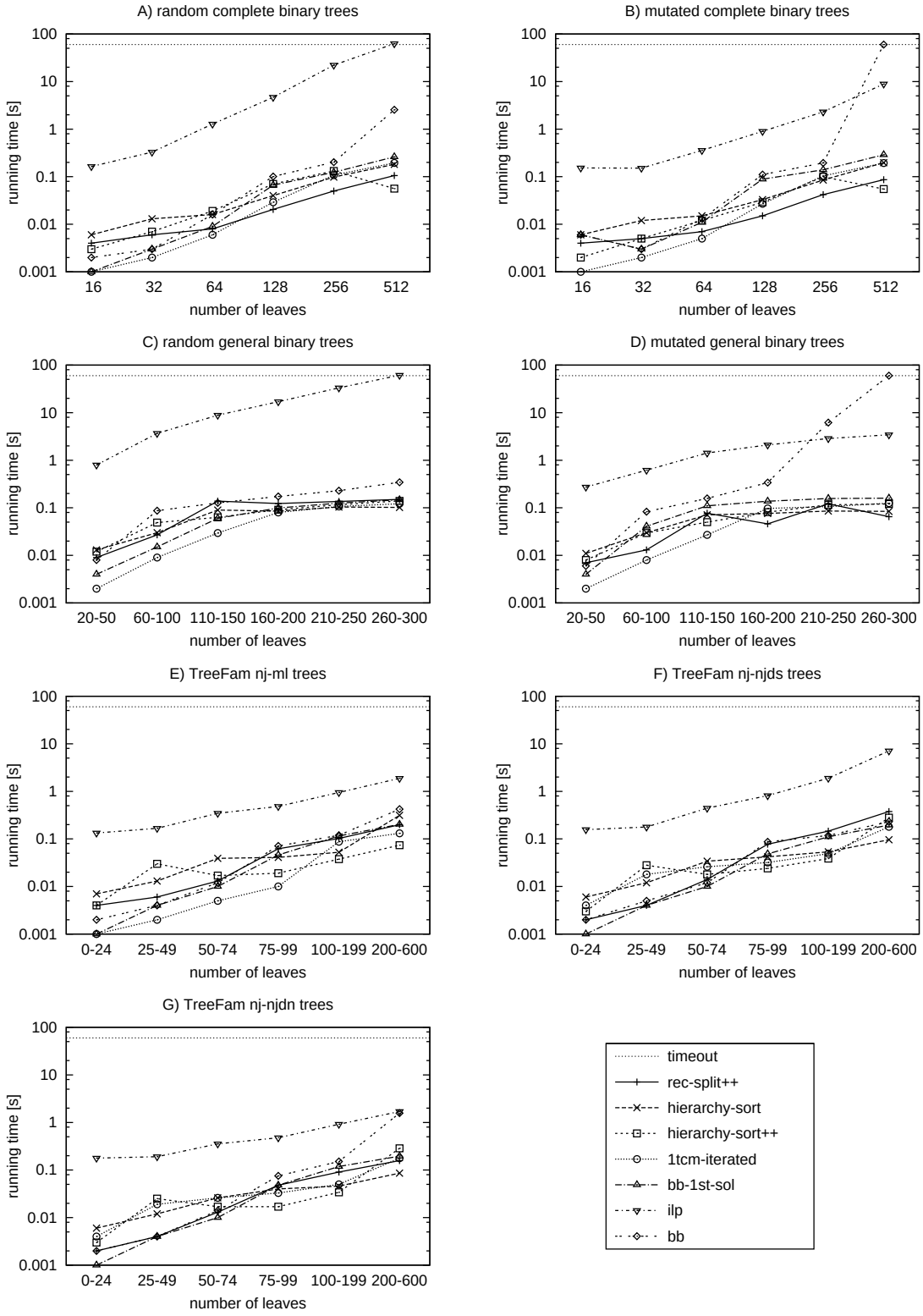


Figure 7.17: Median running times of the algorithms *rec-split++*, *hierarchy-sort*, *hierarchy-sort++*, *1tcm-iterated*, *greedy*, *ilp*, and *bb-exact* (in seconds) for random (A–D) and real-world (E–G) binary tanglegrams.

$O(n^2)$ although *bb-exact*, from which it is derived, has an exponential worst-case running time. From the remaining heuristics, *1tcm-iterated* is fastest for small instances, while *rec-split++* and the two hierarchy-sort variants are faster for larger instances. Recall that our implementation of *1tcm-iterated* runs in $O(n^3)$ time while a faster $O(n \log^2 n)$-time implementation is possible [FKP05]. For complete binary trees, *rec-split++* is the fastest algorithm for $n \geq 128$ with one exception. For $n = 512$ *hierarchy-sort++* is the fastest algorithm and takes less time than for $n = 256$ on sets A and B. The reasons for this behavior are unclear. Summarizing the running times of the heuristics it turned out in our experiments that all of them are fast enough to be applied interactively in a tanglegram visualization tool. In particular, this is the case for *greedy*, the best-performing heuristic.

Considering the exact algorithms, note that *ilp* is slower than the five heuristics by a factor between 10 and 100. Still, it is remarkable that it succeeds to find optimal solutions in less than 10 seconds for mutated random trees (sets B and D) and the TreeFam instances. Only random pairs of trees (sets A and C) with many crossings in the optimal solution are challenging and running times reached the timeout with increasing frequency as Figure 7.15 shows. The running times of *bb-exact* behave oppositely. While *bb-exact* is as fast as the heuristics for small values of n, its running time increases quickly for trees that are relatively similar to each other (sets B, D, and G), where *bb-exact* gets slower than *ilp*. Hence the upper bounds used to prune subtrees of the search tree in *bb-exact* seem to be most efficient for instances of unrelated trees with large crossing numbers. For the largest of our instances *bb-exact* does no longer find all optimal solutions within one minute, most notably for sets B and D (see Figure 7.15).

7.5.5 Discussion

The experimental evaluation clearly indicates that our greedy heuristic, which yields the first feasible tanglegram layout obtained in the exhaustive search of the exact branch-and-bound algorithm, is superior to all other heuristics and thus the method of choice for arbitrary binary tanglegrams. It found optimal solutions in the majority of our examples. Its worst-case performance ratio observed in the more than 10,000 examples was 2.24. With a running time of $O(n^2)$ *greedy* also turned out to be fast enough even for the largest tanglegrams of size up to 600 leaves, which took less than half a second to compute. In practice it might be a good idea to continue running *bb-exact* for a pre-specified time after the first solution has been found in order to find an even better (or the optimal) solution. The ILP is faster for large instances than *bb-exact* (if crossing numbers are not too high) and its running time is less susceptible to outliers. Its disadvantage is, however, that our implementation requires a license for the commercial ILP solver CPLEX. Note that real-world tanglegrams often have a crossing-free layout. By construction, the underlying algorithm *bb-exact* guarantees to find a crossing-free layout as the first solution if it exists. Thus *greedy* is optimal in that case. For the special case of complete binary trees, the the heuristic variant *rec-split++* of our 2-approximation algorithm is faster than *greedy* and achieves a similar performance, which makes it a good alternative for that case.

7.6 Concluding Remarks

In this chapter we have comprehensively studied the binary TL problem, a tree visualization problem, which arises frequently in evolutionary biology. From the theoretical perspective

it has been shown that the problem, which is known to be NP-complete [FKP05], remains NP-complete for complete binary trees and that it is hard to approximate for general binary trees under the Unique Games Conjecture. This led us to develop a 2-approximation and a fixed-parameter algorithm for the special case of complete binary trees.

From the practical perspective, it is important to find (nearly) optimal tanglegram layouts for general binary trees quickly. Hence we gave an ILP formulation and an exact branch-and-bound algorithm as well as a fast and effective greedy heuristic that runs in quadratic time. In an extensive experimental study we evaluated our new algorithms and compared them to two previously suggested heuristics. The study showed that our greedy algorithm is clearly ahead of the other heuristics in terms of solution quality; in many cases (and in most of the real-world instances) it even finds an optimal solution. With computation times of less than half a second even for the largest of our test instances this makes it the method of choice for generating tanglegram layouts in real-world settings. Optimal solutions can be computed within one minute for most test instances, either by the ILP or by the branch-and-bound algorithm.

Open problems. At least three ways of generalizing binary TL are interesting for applications. In software analysis, trees are used to represent package-class-method hierarchies or the decomposition of a project into layers, units, and modules. Such trees are usually not binary and it is an interesting question to study the TL problem for trees of arbitrary node degrees.

A different way of generalizing TL is of interest for phylogenetic trees, our main application. If the co-evolution of two sets of related species, for example, a set of hosts and a set of parasites, is to be studied it often occurs that one species may act as the host for multiple parasites or vice versa. This means that the set of inter-tree edges no longer forms a matching but a general bipartite graph. This is in similar to layered graph drawing but with the restriction that the leaf orders must be compatible with the two trees. The heuristics evaluated in this paper all generalize to this setting with minor adjustments. Would this still allow for constant-factor approximations in the case of complete binary trees?

Since tanglegrams with a small crossing number, k, seem to occur often in practice, it would be desirable to have a fixed-parameter algorithm for general (non-binary) trees that is simpler and faster than the $O^\star(1024^k)$-time algorithm of Fernau et al. [FKP05].

A recent eye-tracking study by Huang [Hua08] suggests that edge crossings might not be as bad as it is often assumed. It is rather the crossing angles of edges that strongly influence readability. It turned out that crossings at angles of roughly 90 degrees hardly influenced the subjects' performance while small crossing angles significantly slowed it down. This observation leads to the interesting TL variant where the problem is to minimize the number of "bad" crossings.

Chapter 8

Cover Contact Graphs

In the majority of graph drawing applications and theoretical studies, graphs are typically represented as node-link diagrams, that is, vertices are represented as points, disks or rectangles and edges are represented as arcs connecting their vertex objects. The previous chapters in this dissertation also adhered to the node-link visualization style. There are, however, also less common geometric representation styles of graphs that nonetheless give rise to interesting questions and applications. For example, in an *intersection graph* vertices are represented by geometric objects such as lines, line segments, disks, etc., and each edge is represented by a non-empty intersection of the two objects corresponding to its incident vertices. *Contact graphs* are defined similarly but we require that the interiors of all sets are disjoint and hence edges correspond to "touching" sets. A classical result in graph theory is Koebe's theorem from 1936 that says that every planar graph can be represented as a *coin graph*, that is, a contact graph of disks [Koe36].

On the other hand, geometric covering problems have been extensively studied in computational geometry and geometric optimization. In a covering problem there is a set of geometric objects, mostly points, that need to be covered under certain constraints by other geometric objects, for example by disks. Typical questions are finding a minimum-size object or finding a minimum number of fixed-size objects that cover all points. In this chapter we combine the graph representation problem and the covering problem, that is, we are interested in finding contact graph representations using certain geometric objects (for example, disks) that at the same time cover a set of geometric seed objects (for example, points). More precisely, each seed must be covered by exactly one covering object. The contact graph is in this case called a *cover contact graph*. We ask two questions: "Is there a connected cover contact graph for the given set of seeds?" and "Can a given graph be realized as a cover contact graph on a given set of seeds?". The results in this chapter are based on joint work with Nieves Atienza, Natalia de Castro, Carmen Cortés, Mari Ángeles Garrido, Clara I. Grima, Gregorio Hernández, Alberto Márquez, Auxiliadora Moreno, José Ramon Portillo, Pedro Reyes, Jesús Valenzuela, Maria Trinidas Villar, and Alexander Wolff [AdCC+08].

8.1 Introduction

Alternative geometric representations of graphs have been studied since the time of Koebe's theorem [Koe36, PA95], a beautiful and classical results in graph theory. The theorem, that has long been forgotten and that was rediscovered several times (see the historical survey of

Sachs [Sac94]), says that every planar graph has a representation as a *coin graph*, that is, a contact graph of disks in the plane. In other words, given any planar graph with n vertices, there is a set of n disjoint open disks in the plane that are in one-to-one correspondence to the vertices such that a pair of disks is tangent if and only if the corresponding vertices are adjacent. Conversely, every coin graph is obviously planar since if we connect the centers of touching disks by straight-line edges, no two edges intersect. Collins and Stephenson [CS03] give an efficient algorithm for numerically approximating the radii and locations of the disks of such a representation of a planar graph. Their algorithm relies on an iterative process suggested by Thurston [Thu80].

Since Koebe there has been a lot of work in the graph-drawing community dedicated to the question which planar graphs can be represented as contact or intersections graphs of which geometric object, see Section 8.2 for a summary of results. Intersection graphs, especially of disks, are used as graph models in multiple application areas, for example, wireless communication networks [Hal80, CCJ90].

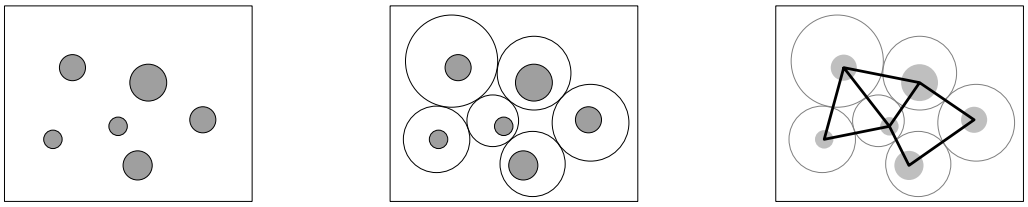
The geometric-optimization community has dedicated a lot of work to the question of how to (optimally) cover geometric objects (usually points) by other geometric objects (like convex shapes, disks, annuli). Several variations of this general problem appear in the literature. Typical objectives are minimizing the (maximum/total) radius of a set of k disks to cover n input points. Alternatively, the disk size might be fixed and the number of disks used to cover the input points is to be minimized. Applications of such covering problems are, for example, geometric facility location problems.

In this chapter we combine the two previous problems: we are given a set of pairwise disjoint geometric objects called *seeds* (such as points or disks) that must be covered by other geometric objects called *covering objects* (such as disks or triangles) whose interiors are disjoint. The covering objects must either represent a given graph or satisfy a given graph property by the way they touch each other. Other than in geometric optimization each of our covering objects must contain exactly one of the seeds. We are not interested in minimizing the sizes of the covering objects or their number; instead we want them to jointly fulfill some graph-theoretic property (like connectivity). Compared to previous work on geometric representation of graphs, the requirement to cover each seed by a covering object significantly restricts the freedom to place the geometric objects.

Model. Given a set $S = \{p_1, p_2, \ldots, p_n\}$ of pairwise disjoint *seeds* in $\mathbb{R}^2$ of some type, a *cover* of S is a set $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$ of closed objects of some type with the property that each object C_i in $\mathcal{C}$ contains exactly one seed s_i in S and that the interiors of no two covering objects in $\mathcal{C}$ intersect. Figure 8.1b depicts a disk cover of the disk seeds in Figure 8.1a. Now the *cover contact graph* (CCG) induced by $\mathcal{C}$ is the contact graph of the elements of $\mathcal{C}$, that is, the graph $G = (V, E)$ with $V = \mathcal{C}$ and $E = \{\{C_i, C_j\} \subseteq \mathcal{C} \mid C_i \neq C_j,\ C_i \cap C_j \neq \emptyset\}$. In other words, two vertices of a CCG are adjacent if the corresponding covering objects touch, that is, their boundaries intersect. Figure 8.1c depicts the CCG induced by the cover in Figure 8.1b. Note that the vertices of the CCG are in one-to-one correspondence to both seeds and covering objects. We consider seeds to be topologically open (except if they are single points). Then seeds can touch each other. Note that we require cover objects to be closed. This makes sure that a cover actually contains a point seed that lies on its boundary. We denote a CCG whose cover consists of disks (triangles) as a disk-CCG (triangle-CCG).

In our work we investigate the following two questions.

Connectivity: Given a set of seeds, does it admit a (1- or 2-) connected CCG?



(a) Disk seeds. (b) Disk cover of (a). (c) CCG induced by (b).

Figure 8.1: Seeds, cover, and CCG.

Realizability: Given a planar graph G and a set of seeds, can G be realized as a CCG on the given seeds?

We also consider an interesting restriction of the above problems where seeds and covering objects must lie in the half plane $\mathbb{R}^2_+$ above and including the x-axis. Seeds are additionally restricted in that each must contain at least one point of the x-axis. In this restricted setting we call the contact graph of a cover a CCG$^+$. See Figures 8.11b and 8.13 in Section 8.4 for examples.

Contributions. In this chapter we consider three classes of seeds: points, disks, and homothetic triangles. (Recall that a homothetic transformation is a dilation followed by a translation.) Our covering objects are either disks or homothetic triangles. First, we consider arbitrary sets of point seeds (Section 8.3). Concerning connectivity we show that we can always cover a set of point seeds using disks or using homothetic triangles such that the resulting CCG is 1- or even 2-connected. Our algorithms run in $O(n \log n)$ worst-case and $O(n^2 \log n)$ expected time, respectively. Concerning realizability we give some necessary conditions and then show that it is NP-hard to decide whether a given graph can be realized as a disk-CCG if the correspondence between graph vertices and point seeds is given. Second, we consider the restriction where we are given a set S of points on the x-axis as seeds (Section 8.4). We show that in this case 1-connectivity is easy: we can realize the cycle C_n as CCG on S and there are trees that can be realized as CCG$^+$ on S. For the case that the correspondence between seeds and vertices is given, we give an algorithm that decides in $O(n \log n)$ time which trees can be realized as CCG$^+$. Third, we consider disk and triangle seeds (Section 8.5). We show that for homothetic triangle seeds on the x-axis, there is always a connected triangle-CCG$^+$ and that for *disk* seeds it is already NP-hard to decide whether they admit a connected disk-CCG.

8.2 Related Work

Intersection and contact graphs. The first class of problems related to cover contact graphs are found in general intersection and contact graphs of geometric objects. The difference to our problem is, however, that the geometric objects are not constrained to cover a set of seeds. Intersection graphs, not necessarily defined geometrically, play an important role in graph theory and are covered in a book by McKee and McMorris [MM99]. In terms of geometric intersection graphs, an analogous result to Koebe's theorem for coin graphs [Koe36] was given by de Fraysseix et al. [dFOdMR94]. They showed that any planar graph has a representation as a contact graph of triangles. The same authors recently

extended this result to planar linear hypergraphs [dFOdMR08], where a hypergraph is linear if any two hyperedges have at most one vertex in common. Badent et al. [BBdG+07] were interested in the question which graphs can be represented as contact graphs of homothetic triangles. There are planar graphs that have no such representation but they proved that two-terminal series-parallel digraphs and *partial planar 3-trees*[1] can be realized as homothetic-triangle contact graphs.

Disk graphs have been studied extensively. Most recognition problems are NP-hard, that is, it is NP-hard to decide whether a given graph can be represented as contact or intersection graph of some type of disks. As already mentioned before, contact graphs of arbitrary disks are exactly the planar graphs [Koe36] (and hence can be recognized in linear time [HT74]) but deciding whether a graph has a representation as the intersection graph of arbitrary disks in the plane is NP-hard as showed by Hliněný and Kratochvíl [HK01]. Breu and Kirkpatrick showed that for unit disks the recognition problem is NP-hard both for intersection and for contact graphs [BK96, BK98]. The NP-hardness results can be extended to *bounded-ratio* intersection and contact graphs of disks [BK96], where a bounded-ratio disk graph is a disk graph with disk radii in the range $[1, r]$ for some fixed $r \geq 1$. In higher dimensions, Hliněný and Kratochvíl showed that the recognition problem is NP-hard for unit-ball contact graphs in dimensions 3, 4, 8, and 24 [HK01]. They conjecture that this holds for all dimensions $d \geq 2$. Unit-ball intersection graphs are NP-hard to recognize in two and three dimensions and it was conjectured that this holds for any dimension $d \geq 2$ as well [BK98].

Another class of graphs that has attracted a lot of attention is the class of curve intersection graphs (also called *string graphs*) [EET76]. An interesting subclass are segment intersection graphs [KM94]. It is a famous open question asked by Scheinerman in 1984 [Sch84] whether every planar graph is the intersection graph of segments in the plane. As a recent example, de Fraysseix and Ossona de Mendez [dFOdM07] showed that any four-colored planar graph without an induced four-colored cycle C_4 is the intersection graph of a family of line segments. In terms of curve contact graphs, de Fraysseix et al. [dFOdMP91] showed that every planar bipartite graph is the contact graph of a family of horizontal and vertical segments. De Castro et al. [dCCDM02] extended this result by showing that every planar triangle-free graph can be represented as a segment contact graph with only three segment directions. On the other hand Hliněný [Hli01] showed that the recognition problem of segment contact graphs is NP-complete. The PhD thesis of Hliněný [Hli00] is a good starting point for further literature research on curve contact graphs.

Covering problems. The second class of problems related to cover contact graphs are geometric covering problems. The difference is now that in these problems we do not have the restriction that each seed must be covered by exactly one covering object and that the contact graph of the covering objects must satisfy some graph-theoretic property. As an example take Welzl's famous randomized algorithm for finding the smallest enclosing ball of a set of points [Wel91]. Another example is the widely-applicable *shifting technique* by Hochbaum and Maass that yields polynomial-time approximation schemes, for example, for covering points with a minimum number of unit disks or rectangles [HM85]. An approximation scheme for the geometric k-median problem, where n points need to be

[1] A planar 3-tree is a recursively defined graph. The complete graph on three vertices is a planar 3-tree. Any graph that is obtained from a planar 3-tree by adding to it a new vertex inside a triangular face and connecting it to the three vertices of this face is again a planar 3-tree. A *partial* planar 3-tree is a subgraph of a planar 3-tree.

covered by k variable-size disks, is given by Arora [Aro03], for further approximation results see also [Vaz01, Chapter 25]. As a last example, Clarkson and Varadarajan [CV05] give polynomial-time approximation algorithms for several variations of geometric set cover problems, for example, covering fat triangles by pseudodisks or by other fat objects.

Combining contact graphs and covering problems. Abellanas et al. [ABH+06] proved that the following packing problem, which they call the *coin placement problem*, is NP-complete. Given n disks of varying radii and n points in the plane, is there a way to place the disks such that each disk is centered at one of the given points and no two disks overlap?

Abellanas et al. [AdCH+06] considered a problem that is closely related to cover contact graphs. They showed that given a set of points in the plane, it is NP-complete to decide whether there are disjoint disks centered at the points such that the contact graph of the disks is connected. The difference to our problem definition is that we require only that each disk covers a point, but the disk does not need to be centered at that point.

Given a pair of touching (convex) covering objects, we can draw the corresponding edge in the CCG by a two-segment polygonal line that connects the incident seeds and uses the contact point of the covering objects as bend. This is a link to the problem of point-set embeddability. We say that a planar graph G is *k-bend (point-set) embeddable* if for any point set $P \subset \mathbb{R}^2$ there is a one-to-one correspondence between V and P such that the edges of G can be drawn as non-crossing polygonal lines with at most k bends per edge. Kaufmann and Wiese [KW02] showed that (a) every 4-connected planar graph is 1-bend embeddable, (b) every planar graph is 2-bend embeddable, and (c) given a planar graph $G = (V, E)$ and a set P of n points on a line, it is NP-complete to decide whether G has a 1-bend embedding that maps V one-to-one on P.

For the case of a given correspondence between vertices and points (which is relevant to this paper) Pach and Wenger [PW01] showed how to compute in $O(n^2)$ time a plane embedding of the given graph where each vertex is represented by the corresponding point and each edge is represented by a polygonal line with $O(n)$ bends. They also showed that $\Omega(n)$ bends are needed in the worst case.

8.3 Point Seeds in the Plane

In this section we study point seeds which may take arbitrary positions in the plane. If not stated otherwise our results hold for both disk covers and (homothetic) triangle covers. We start with the connectivity problem.

8.3.1 Connectivity

It is known to be NP-hard to decide whether a given set of points can be covered by a set of pairwise disjoint open disks, each centered on a point, such that the contact graph of the disks is connected [AdCH+06]. In contrast to that result we give a simple sweep-line algorithm that covers point seeds by (non-centered) disks such that their contact graph is connected.

Theorem 8.1 *Every set S of n point seeds has a connected CCG. Such a CCG can be constructed in $O(n \log n)$ time and linear space.*

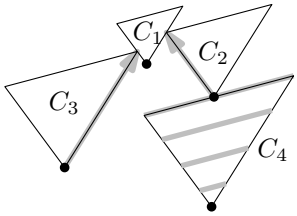


Figure 8.2: Constructing a connected CCG by inflating triangles as covering objects. Three contact types are possible: the top edge of the new triangle touches the bottom vertex of a previous triangle (C_4), the top right vertex touches a left edge (C_3), or the top left vertex touches a right edge (C_2).

Proof. We first sort the given seed set S by non-increasing ordinate. Let $p_1, \ldots, p_n$ be the resulting sequence of points. Then we proceed incrementally from top to bottom. We cover p_1 by a covering object C_1 (disk or triangle, depending on the case) of some fixed size such that p_1 is the bottommost point of C_1. Now assume that $k > 1$ and that the $k-1$ topmost points have been covered such that the contact graph of their cover is connected. Then we cover $p_k = (x_k, y_k)$ initially by an infinitesimally small covering object C_k and inflate C_k with p as the bottommost point until C_k eventually touches one of the previously placed covering objects. Figure 8.2 shows an example with homothetic triangles as covering objects.

It is possible to do this construction by incrementally building an abstract Voronoi diagram with respect to accordingly defined bisectors. It is not clear, however, how to implement this in $O(n \log n)$ time. Instead we use the following simpler algorithms for triangles and disks, respectively.

For triangles it is easy to determine the size of the k-th triangle C_k given that triangles $C_1, \ldots, C_{k-1}$ have been placed. The idea is that we maintain three data structures, one for each type of collision during the afore-mentioned inflation step, see Figure 8.2. We first precompute, in $O(n \log n)$ time and linear space, a data structure for so-called *segment-dragging queries* [Mit92]. Given a fixed direction d and a wedge W with apex $(0, 0)$, a data structure for segment dragging yields, in $O(\log n)$ time, for a given query point $p \in \mathbb{R}^2$ the first point hit by a line segment of direction d whose endpoints run from p along the edges of the wedge $W + p$ with apex p. This data structure gives us seeds that are hit by (the relative interior of) the top edge of C_k (see C_4 in Figure 8.2). It remains to determine the first triangles that are hit by rays on which the top left and top right vertex of C_k travel when we inflate C_k. This can be done by two simple ray-shooting data structures. Both data structures can be implemented by dynamic balanced binary search trees, since in each of them all query rays have the same direction, and so do the segments that are potentially hit. Thus we again have a query time of $O(\log n)$. Once a new triangle is added to the cover, we insert the new left and right triangle edges dynamically to the search trees in $O(\log n)$ time per piece. Thus our algorithm for connected triangle-CCGs runs in $O(n \log n)$ time and needs linear space.

For disks we do a top-to-bottom sweep with a horizontal sweep line $\ell : y = c$. As before, let $C_1, \ldots, C_{k-1}$ be the disks that have already been placed, that is, whose south poles lie above ℓ. We maintain the lower envelope of functions $f_1^c, \ldots, f_{k-1}^c$, where f_j^c is the locus of the centers of all disks that touch both disk C_j and the sweep line. It is easy to see that f_j^c is the parabola whose focus is the center of C_j and whose directrix is the horizontal line with y-coordinate $c - r_j$, see Figure 8.3. Note that given a query point (x, c) on the sweep line, the largest disk with south pole (x, c) that does not intersect the interior of any

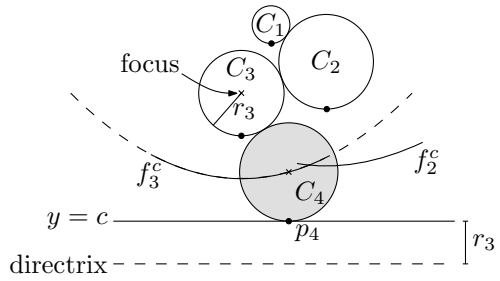


Figure 8.3: Finding the covering disk C_4 for seed p_4 that touches a previous disk. The center of disk C_k lies on the lower envelope of the functions f_j^c for $j \leq k-1$. The focus and directrix for parabola f_3^c are shown.

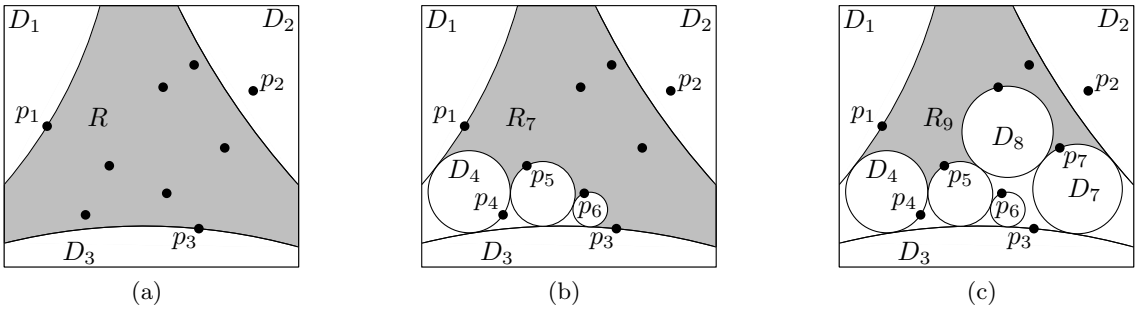


Figure 8.4: Three steps in the construction of a biconnected disk-CCG.

previously placed disk has center $(x, \min_j f_j^c(x))$ and radius $\min_j f_j^c(x) - c$. This disk can easily be computed given the lower envelope.

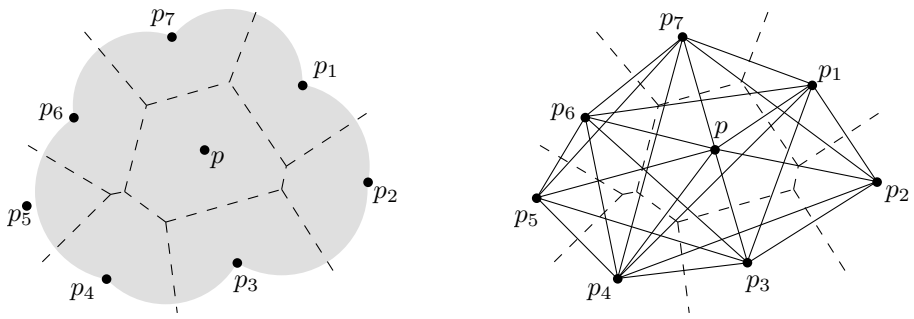
Our sweep is very similar to Fortune's sweep [For86] for computing the Voronoi diagram of weighted points (or disks). The only difference is that we do not know the weight of a point p beforehand; we compute the weight of p (by querying the lower envelope) as soon as we reach p. The handling of changes in the lower envelope and the insertion of new parabolas are essentially the same as in Fortune's sweep. Thus the running time of $O(n \log n)$ and the linear space consumption carry over. □

In fact, even a biconnected CCG for any set of n point seeds exists as the following theorem assures.

Theorem 8.2 *Any set S of n point seeds has a biconnected CCG. Such a CCG can be constructed in $O(n^2 \log n)$ expected time using linear space.*

Proof. We first consider disks as covering objects. Let D_1, D_2, and D_3 be three congruent disks that touch each other. They delimit a pseudo-triangular shape R. Choose the three disks such that each disk D_i contains a unique point $p_i \in S$ and such that $S \backslash \{p_1, p_2, p_3\} \subset R$, see Figure 8.4a.

In order to cover the remaining points we assume that disks $D_4, \ldots, D_{i-1}$ have been placed such that each covers a unique point of S and touches two previously placed disks, see Figure 8.4b. Thus the contact graph of $D_1, \ldots, D_{i-1}$ is biconnected. Let R_j be a connected component of $R \setminus \bigcup_{j=4}^{i-1} D_i$ that contains at least one uncovered point. Use Fortune's sweep [For86] to compute the combined Voronoi diagram of the disks incident to R_j and the points in $S \cap R_j$. This takes $O(n \log n)$ time and the resulting Voronoi diagram



(a) Influence area of seed p.

(b) Hyperinfluence graph $HI(S)$.

Figure 8.5: Influence area and hyperinfluence graph for seeds S.

has complexity $O(n)$. The part of the Voronoi diagram in R_j is the locus of the centers of all disks that lie in R_j and touch $\partial R_j \cup (S \cap R_j)$ in at least two points, where ∂R_j is the boundary of R_j.

Now we make a simple but crucial observation: if D is a disk that (a) lies in R_j, (b) contains a seed $s \in S \cap R_j$ on its boundary, and (c) touches two of the previous disks, then D is centered at a *vertex* of the Voronoi diagram. Thus a disk $D^\star$ fulfilling (a)–(c) can be found in linear time and, by construction, does not contain any point of S in its interior. (If by any chance all such disks touch more than one point of S, we re-start the whole computation with three slightly wiggled initial disks D_1, D_2, and D_3. Then the probability of this degeneracy becomes 0.) Now set $D_i = D^\star$, and repeat the process until all seeds are covered. This takes $O(n^2 \log n)$ time in total.

The case of triangles can be handled analogously. Choosing any reference point in the triangular shape, a structure similar to the medial axis can be computed in $O(n \log n)$ time and updated in $O(n)$ time in each of the $n - 3$ phases. □

8.3.2 Realizability

In this section we first give two necessary conditions that a planar graph must satisfy in order to be realizable as a disk-CCG on a given seed set. We show that there is a plane geometric graph on six vertices that cannot be represented as disk-CCG. Finally we investigate the complexity of deciding realizability.

To formulate our necessary conditions for realizability we define the *hyperinfluence graph* on the given seed set S. This graph is inspired by the sphere-of-influence graph defined by Toussaint [Tou88] (see also [HJLM93, JLM95] for more results on sphere-of-influence graphs). So given a seed set S and a point $p \in S$ let the *influence area* of p be the closure of the union of all empty open disks D (that is, $D \cap S = \emptyset$) that are centered at vertices of the Voronoi region of p, see Figure 8.5a. We call the intersection graph of the influence areas of all seeds in S the *hyperinfluence graph* of S and denote it by $HI(S)$, see Figure 8.5b.

Proposition 8.1 *Let S be a set of point seeds and let G be a graph realizable as a disk-CCG on S. Then*

(i) G is a subgraph of $HI(S)$, and

(ii) G has a plane drawing where each vertex is mapped to a unique point in S and each edge is drawn as a polygonal line with at most two segments (that is, with at most one bend per edge).

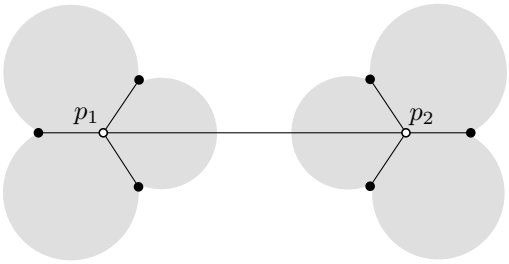


Figure 8.6: A non-realizable graph. The influence areas of p_1 and p_2 do not intersect and thus no two covering disks of p_1 and p_2 can touch.

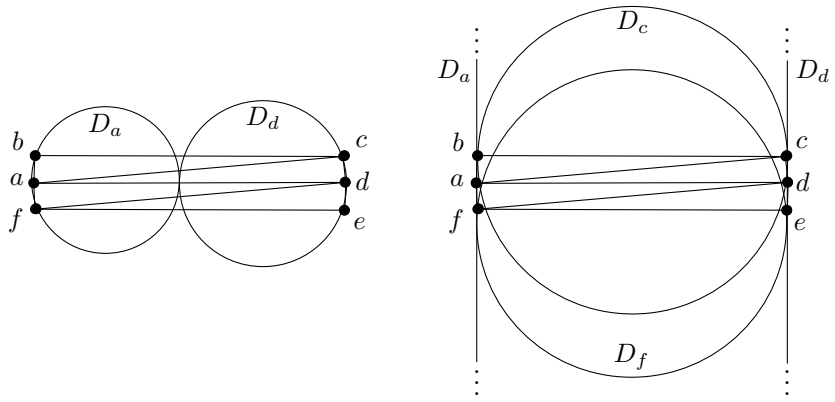


Figure 8.7: Non-realizable Delaunay triangulation of six points in convex position.

Proof. Both properties are straightforward to obtain. Property (i) is based on the observation that any possible covering disk of p is contained in the influence area of p. Thus, if the covering disks of two seeds are in contact, their influence areas intersect. Property (ii) is obtained by representing each edge of the CCG by two line segments that connect the adjacent seeds with the point of tangency of their covering disks. □

While it is NP-complete to verify property (ii) of Proposition 8.1 even if all seeds lie on a line [KW02], property (i) of Proposition 8.1 gives us a way to show non-realizability of certain geometric graphs, for example, the graph depicted in Figure 8.6. The edge p_1p_2 of the graph cannot be realized in a CCG with the given seeds, because the shaded influence areas of p_1 and p_2 do not intersect. This graph is thus an example of a non-realizable graph with eight vertices. On the other hand it is easy to see that any three-vertex graph can be realized on any three-point seed set. Now it is interesting to ask for the least n for which there is a plane n-vertex geometric graph G that cannot be realized as CCG.

Proposition 8.2 *There is a set S of six point seeds in convex position such that their Delaunay triangulation is not representable as a CCG.*

Proof. Let $S = \{a, b, c, d, e, f\}$ be six points in convex position that are connected by the edges of their Delaunay triangulation as shown in Figure 8.7. Since the points a and d are connected, the covering disks D_a and D_d of the points a and d must touch each other in one of two ways. Either the tangent point of the disks lies inside the convex hull of S (left part of Figure 8.7), or D_a and D_d are very large and lie to the left of a and to the right of d, in which case they touch far above or below S as indicated in the right part

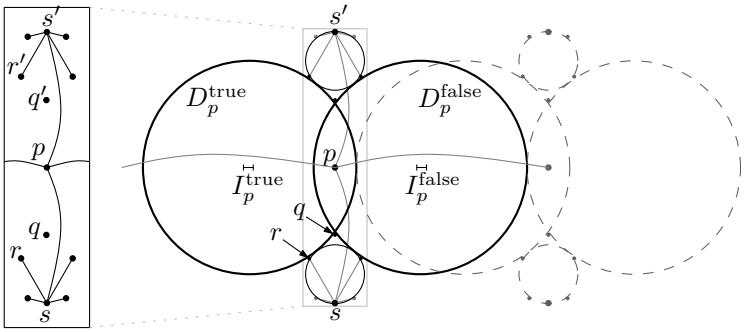


Figure 8.8: Two chain links (solid and dashed) and the graph structure around p.

of Figure 8.7. In the first case we cannot find a disk that covers c and touches D_a. In the second case we can assume that the boundaries of D_a and D_d are two almost parallel lines in the vicinity of the six points. The disks D_c and D_f covering c and f must both touch D_a and D_d. But if c and f are close enough to a and d then D_c and D_f cannot be disjoint. □

So we have seen that there are pairs of (quite small) graphs and seed sets such that the graph cannot be realized on the seed set as disk-CCG. A natural question to ask is whether a given graph is realizable as CCG on a given seed set or not. Of course Koebe's theorem [Koe36] guarantees that for any planar graph G we can find a seed set S such that it is possible to realize G on S. However, if the seeds and the vertex–seed correspondence are given, the problem becomes NP-hard as the next theorem shows.

Theorem 8.3 *Given a set S of points in the plane and a planar graph $G = (V, E)$ with a bijection between V and S, it is NP-hard to decide whether G is realizable as disk-CCG on S.*

Proof. We show the NP-hardness by reduction from the NP-complete Planar3-Sat problem [Lic82] that has been defined in Section 2.3. Note that instead of the orthogonal layout of the variable-clause graph H_φ (recall Figure 2.3), we use a slanted layout of H_φ, where all angles are multiples of 60 degrees; this is similar to a reduction by Cabello et al. [CDR04].

Next, we construct gadgets for variables and clauses as seeds and edges on a triangular grid such that the resulting graph can be realized as CCG if and only if the Boolean formula φ is satisfiable. One basic ingredient are chains of linearly connected points such that each point can be covered only by two combinatorially different disks depending on the truth value encoded by the respective chain structure. These chains will be used for the variables and for the literal connections to the clauses. The structure of the chains is exemplified in Figure 8.8 which shows two chain links and the underlying graph. The two desired covering disks D_p^{true} and D_p^{false} of the central point seed p are drawn in solid black. They both touch the covering disks of the two *stopper elements* s and s' above and below p as required by the edges ps and ps'. We explain the function of a stopper element by looking at s. The positions of the singleton seed q and the seed r, which is connected to s by an edge, induce that the centers of all valid covering disks to the left of p must lie on the line segment I_p^{true}, which can be made arbitrarily short by slightly shifting the seeds of the stopper element. The same holds for I_p^{false} and disks to the right of p.

Variable gadgets. Putting chain links together such that they must touch their neighbors we construct variable gadgets as shown in Figure 8.9. Each variable is represented by a chain of pairs of successive left and right turns of 60 degrees. Note that the turns of 60 degrees adhere to the grid. Once the covering disk of the first seed is fixed to one of the two possibilities *all* successive disks in that chain become fixed because they have to touch their predecessor as well as their two stopper elements. We define the black variable configuration in Figure 8.9a as *true* and the gray one as *false*. At the bends of the variable gadget literal chains can connect from above or below as depicted. If a literal has the value *false* the covering disks of the literal chain are pulled towards the variable (see the black configuration of the lower literal chain). Otherwise both configurations of the literal chain are possible and we may choose the one where the covering disks are pushed away from the variable chain (see the black configuration of the upper literal chain). Figure 8.9b shows two close-up views of the truth value transfer for a negated literal connecting to the variable gadget in its *true* state. (Note that for positive literals the three special seeds at the end of the literal chain are mirrored at the axis of the literal chain.) The left-hand side of Figure 8.9b is an invalid configuration because one of the final disks of the literal chain intersects a covering disk of the variable gadget. Only the configuration on the right-hand side, where the covering disks are pulled towards the variable gadget and hence encode the correct value *false*, is valid.

Clause gadgets. Figure 8.10 depicts the clause gadget. The three literal chains are meeting symmetrically at angles of 120 degrees each. At the end of each chain there is a triangle of three seeds whose covering disks need to touch each other pairwisely. If the literal evaluates to *true*, that is, the covering disks of the literal chain are pushed towards the clause gadget, the last disk of the chain is able to touch the disks of the other two seeds in the vicinity of their position. In that case the overall area used by the three disks of the triangle is relatively small (see the literal coming from the right in Figure 8.9a). If, on the other hand, the literal evaluates to *false*, the disks of the literal chains are pulled towards the variable gadget. Hence the last disk of the chain is pulled away from the other two seeds of the triangle; these two disks, consequently, need to grow strongly. Figure 8.10 shows that two false literals can still be accommodated (Figure 8.9a) while three false literals clearly cannot (Figure 8.9b).

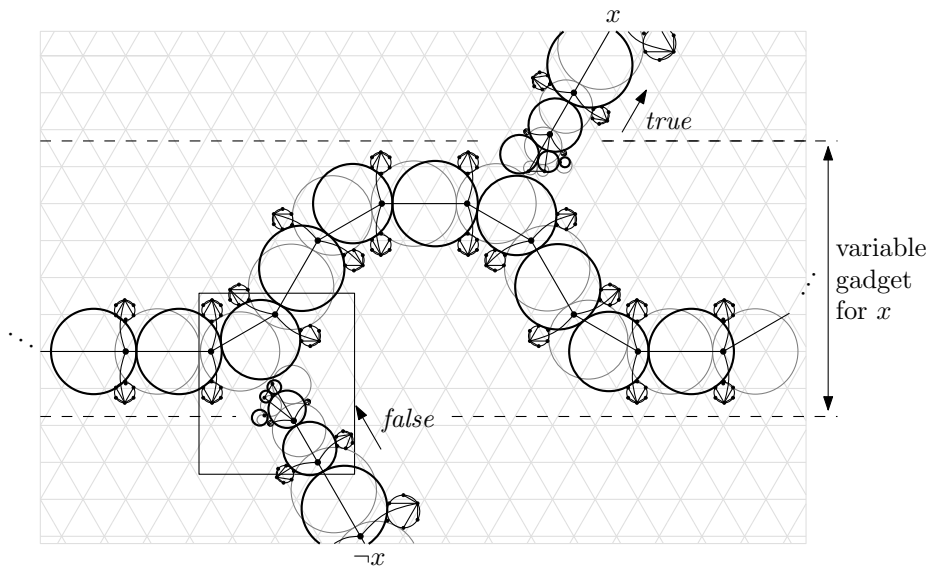
Reduction. From the construction of the gadgets we have the following:

(i) Each variable gadget realizes its respective subgraph by one of two combinatorially different configurations, which represent the truth values *true* and *false*.

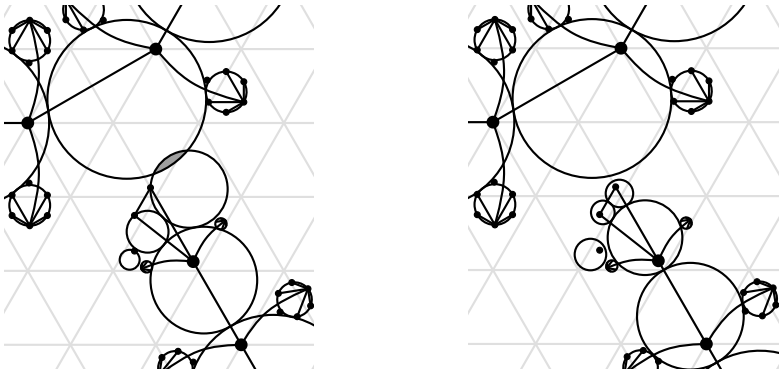
(ii) Each literal gadget similarly realizes its subgraph in one of two configurations; if the literal evaluates to *false*, only one configuration is possible (otherwise two disks would intersect in their interior), if the literal evaluates to *true*, both options are possible.

(iii) Each clause gadget is a space-restricted area that can accommodate the covering disks of at most two false literal gadgets, otherwise there will be some disk overlap.

This concludes the proof of the reduction since the constructed graph can be realized on the constructed seeds if and only if the corresponding planar Boolean 3-Sat formula is satisfiable. We embed all seeds on a hexagonal grid. This grid has polynomial size since the variable-clause graph of φ is embeddable on a grid whose size is quadratic in the length of φ. Hence the reduction takes polynomial time. □



(a) Gadget for a variable x in *true* state (black cover) and in *false* state (gray cover).



(b) Close-up view of the value transfer from variable chain to literal chain (see selection box in subfigure (a)). The left configuration has an invalid disk overlap; the right configuration is valid.

Figure 8.9: Variable gadget.

8.4 Point Seeds on a Line

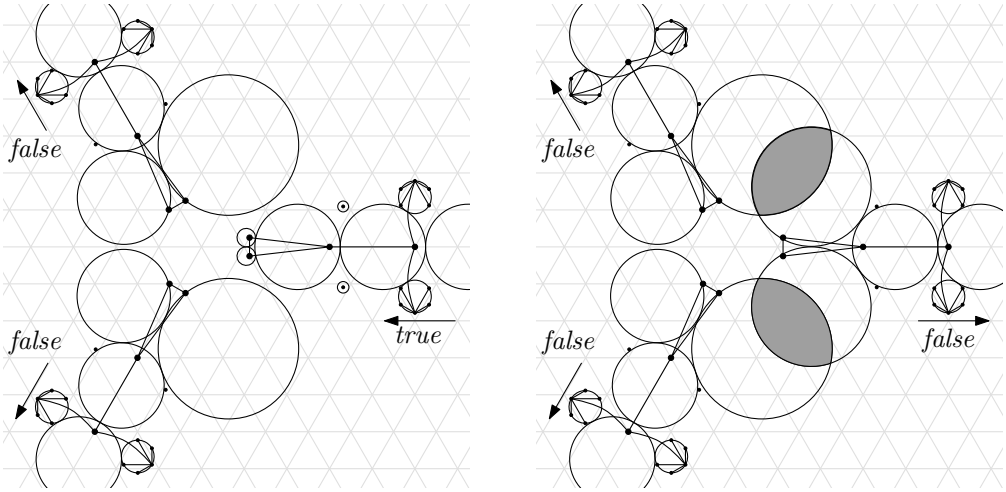
In this section, we restrict the seeds to be points on the x-axis and consider covers in the CCG as well as in the CCG$^+$ setting. Since the connectivity question is answered by some of our realizability results, we focus on the latter problem.

8.4.1 Realizability

We consider the following four questions. Note that seeds now correspond to real numbers, so we can use the natural order $<$ in $\mathbb{R}$ to compare them. All covers consist of disks unless stated otherwise (for example, in Q4).

Q1. Given a graph class $\mathcal{C}$ (for example, the class of trees), does it hold that for any seed set S there is a graph in $\mathcal{C}$ that is realizable as CCG or CCG$^+$ on S?

We show: This is true for the combinations (cycles, CCG) and (trees, CCG$^+$).



(a) Clause evaluates to *true*. (b) Clause evaluates to *false*.

Figure 8.10: Clause gadget.

Q2. Given a graph class $\mathcal{C}$, does it hold that for any graph G in $\mathcal{C}$ there is a seed set S such that G can be realized as CCG or CCG$^+$ on S?

We show: This is true for the combination (trees, CCG$^+$).

Q3. Let $\mathcal{C}$ be a fixed graph class. Given a graph $G \in \mathcal{C}$ with a labeling $\lambda : V \to \{1, \ldots, n\}$ of its vertices, is there a sequence $s_1 < \cdots < s_n$ of seeds on the x-axis and a realization of G as CCG or CCG$^+$that maps each vertex v to the corresponding seed $s_{\lambda(v)}$?

We show: There is an $O(n \log n)$ decision algorithm for the combination (trees, CCG$^+$).

Q4. Let $\mathcal{C}$ be a fixed graph class. Given a seed set S and a graph $G = (V, E) \in \mathcal{C}$ with a one-to-one correspondence between S and V, can G be realized on S as triangle-CCG or triangle-CCG$^+$?

We show: There is an $O(n \log n)$-time decision algorithm for (trees, triangle-CCG$^+$).

Note that the above questions require more and more concrete information about the seed set, ranging from no information (Q2) via a fixed order (Q3) to complete information (Q4).

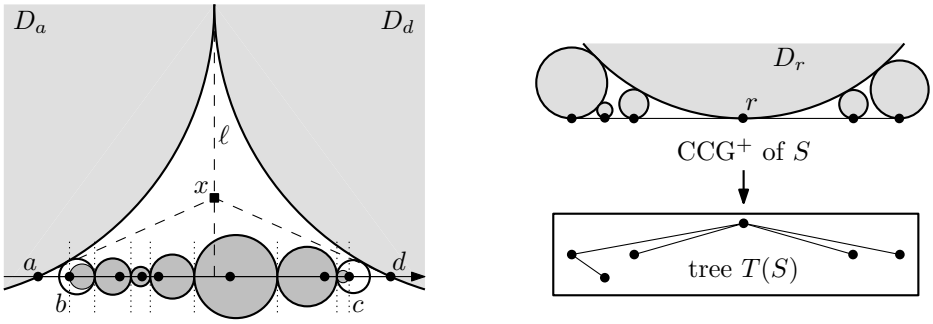
We start with question Q1.

Proposition 8.3 *Let S be a set of n point seeds on a line, then*

(i) the n-vertex cycle C_n can be realized as CCG on S, and

(ii) there is a tree $T(S)$ that can be realized as CCG$^+$ on S.

Proof. (i) Let S be ordered from left to right and let a, b, c, and d be the first, second, second last and last point in S, see Figure 8.11a. Consider the one-dimensional Voronoi diagram of S. We shift the first Voronoi point between a and b to b and the last point between c and d to c. The resulting points are marked by vertical dotted segments in Figure 8.11a. Each cell of the resulting diagram is a segment of the x-axis and contains a seed. Cover the seed by a disk whose diameter is the segment. The resulting disks are shaded in dark gray in Figure 8.11a. Clearly each seed in $S \setminus \{a, d\}$ is now covered by a



(a) C_n is realizable as CCG. (b) Tree $T(S)$ is realizable as CCG$^+$.

Figure 8.11: Graphs that can be realized on a given one-dimensional n-point seed set S.

disk that touches the disk of its predecessor and the disk of its successor (where those exist). Choose a point x on the perpendicular bisector ℓ of a and d such that all (closed) disks lie in the interior of quadrangle $(a, x, d, -x)$. Cover a and d by disks D_a and D_d, that touch the quadrangle in a and d, respectively, and touch each other on ℓ. Clearly these two disks (light gray in Figure 8.11a) do not touch any of the other $n-2$ disks yet. So finally we enlarge the disk D_b of b by moving the left endpoint of its diameter towards a until D_b touches D_a. The disk D_c of c is enlarged analogously towards d. This closes the cycle.

A similar construction that first connects the points from b to c by a path and then closes the cycle from a to d can be used for homothetic triangles as covering objects.

(ii) We pick any seed r as root and cover it by a disk D_r whose projection on the x-axis spans all seeds. Then we grow a disk from each seed until it touches one of the previously placed disks, see Figure 8.11b. A cycle can appear only if a new disk accidentally touches more than one previously placed disk. In this case we increase the radius of D_r by a randomly chosen $\varepsilon > 0$ and repeat the process. Then the probability of constructing a tree is 1.

For triangular covering objects we start with a large triangle placed with its bottommost vertex at the leftmost seed such that its projection on the x-axis contains all seeds (apply the same idea from right to left if the triangle is tilted to the left). Now we iteratively grow a triangle from the next seed until it touches one of the earlier triangles with its top left vertex. Note that the top right vertex can never touch a previous triangle and hence the CCG$^+$ obtained is a tree. □

In terms of this chapter, a coin graph is obtained when seeds are points and covering objects are disks centered at seeds. Thus Koebe's theorem establishes that it is always possible to choose seeds in the plane such that any given planar graph is realizable as a coin graph on them. We have seen in Proposition 8.3 that C_n is realizable as a CCG on any seed set on a line. One can ask whether a Koebe-type theorem also holds in this restricted setting. Kaufmann and Wiese [KW02] have shown, however, that there is a plane triangulated 12-vertex graph (see Figure 8.12) that cannot be drawn with only one bend per edge if vertices are restricted to a line. Now Proposition 8.1 (ii) implies that this 12-vertex graph is not realizable as CCG if the seeds lie on a line. On the positive side, we can show that a Koebe-type theorem holds for the combination (trees, CCG$^+$). This is an answer to Q2 and in a way dual to Proposition 8.3 (ii).

Proposition 8.4 *For any tree T there is a seed set $S(T) \subset \mathbb{R}^1$ such that T is realizable as CCG$^+$ on $S(T)$.*

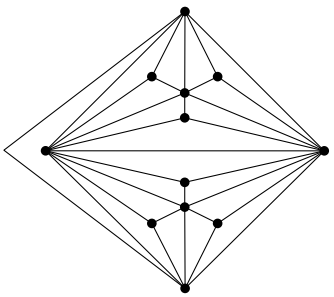


Figure 8.12: Graph by Kaufmann and Wiese [KW02].

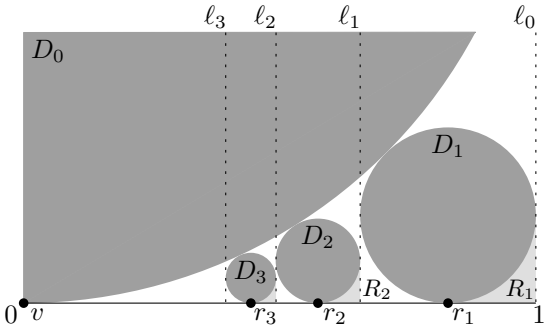


Figure 8.13: Constructing a seed set $S(T)$.

Proof. Our construction is recursive. We traverse the vertices of T in breadth-first order. We map the root v of T to 0 and cover it by a disk D_0 of radius 1, see Figure 8.13. If v has no children, we are done. Otherwise let $r_1, \ldots, r_k$ be the children of v. Define ℓ_0 to be the line $x = 1$. Note that ℓ_0 touches D_0. Now for each child r_i of v do the following. Place the largest disk D_i that fits into the region delimited by D_0, the x-axis, and the line ℓ_{i-1}. Map r_i to the south pole of D_i, that is, the point where D_i touches the x-axis. Note that D_i also touches ℓ_{i-1} on its right-hand side. Define ℓ_i to be the left vertical that touches D_i. Figure 8.13 shows the placement of the seeds for the three children r_1, r_2, r_3 of v and their covering disks D_1, D_2, D_3.

It is not hard to see that we can place an arbitrary number of children of v in this way. Clearly the disks of any two children of v are disjoint. Note that all these disks lie in the region R_0 delimited by D_0, ℓ_0, and the x-axis. For $i = 1, \ldots, k$ we define the region R_i delimited by D_i, ℓ_{i-1} and the x-axis. These regions are shaded in light gray in Figure 8.13. Each of them is congruent to R_0. Thus we can repeat the process for placing the children of v in R_0 for the children of $r_1, \ldots, r_k$ in the respective regions $R_1, \ldots, R_k$. These regions are pairwise disjoint, so the disks of two different grandchildren of v do not intersect.

The same idea can also be used to show the proposition for homothetic triangles as covering objects. □

In Proposition 8.4 above, we had complete freedom to choose the seeds. Now we turn to question Q3, where we are not just given a tree, but also an order of its vertices that must be respected by the corresponding seeds. Kaufmann and Wiese [KW02] have investigated a related problem. They showed that it is NP-complete to decide whether the vertices of a given (planar) graph can be put into one-to-one correspondence with a given set of points on a line such that there is a plane drawing of the graph with at most one bend per edge. We call such a drawing a 1d-1BD. If additionally all bends lie on one side of the line, we call the drawing a 1d-1BD$^+$. Note that a 1d-1BD of a graph G can be seen as a two-page book embedding [CLR87], where the edges drawn below the line that contains the vertices (called the *spine* in book embeddings) correspond to edges on one page while the edges above the spine correspond to edges on a second page. Similarly, a 1d-1BD$^+$ of G can be considered as a one-page book embedding.

Note that the hardness result of Kaufmann and Wiese does not yield the hardness of the one-dimensional CCG realizability problem, since not every graph that can be one-bend embedded on a set of points on a line is realizable as CCG, let alone as CCG$^+$. Our next result explores the gap between Kaufmann and Wiese's one-dimensional embeddability

problem and the situation in Proposition 8.4.

More formally, given an n-vertex tree T and a (bijective) labeling $\lambda : V \to \{1, \ldots, n\}$ of its vertices, we say that T is λ-*realizable* (as CCG, CCG$^+$, 1d-1BD, 1d-1BD$^+$) if there is a sequence $s_1 < \cdots < s_n$ of seeds in $\mathbb{R}^1$ and a realization of T (as CCG, CCG$^+$, 1d-1BD, 1d-1BD$^+$) that maps each vertex v to the corresponding seed $s_{\lambda(v)}$.

In order to obtain a characterization of trees that are λ-realizable as CCG$^+$, we need the following definition. Given a graph $G = (V, E)$ with vertex labeling λ, a *forbidden pair* is a pair of edges $\{ab, cd\}$ such that $\lambda(a) < \lambda(c) < \lambda(b) < \lambda(d)$. Note that it is impossible to embed the edges of a forbidden pair simultaneously above the x-axis.

Theorem 8.4 *For a λ-labeled tree T the following statements are equivalent:*

(i) T is λ-realizable as a CCG$^+$.

(ii) T is λ-realizable as a 1d-1BD$^+$.

(iii) T does not contain any forbidden pair.

Proof. We first show that (i) and (ii) and then that (ii) and (iii) are equivalent.

(i) $\Rightarrow$ (ii): Given a λ-realization of T as CCG$^+$ on some seed set $S \subset \mathbb{R}^1$, we can use the idea of Proposition 8.1 (ii) to get a one-bend embedding in $\mathbb{R}^2_+$ on the same seed set by drawing each edge as the two segment polyline from the first seed via the point of tangency of the corresponding covering disks to the second seed.

(ii) $\Rightarrow$ (i): Now we are given a λ-realization of T as 1d-1BD$^+$ on some seed set $S \subset \mathbb{R}^1$. Let v be a *free* vertex of T if T has no edge uw with $\lambda(u) < \lambda(v) < \lambda(w)$. Note that T has at least two free vertices, namely those with λ-values 1 and n. Pick any free vertex v as root of T. Represent v by a seed at the origin and the unique unit disk D_0 in $\mathbb{R}^2_+$ touching the origin. The root v partitions $T \setminus \{v\}$ in two (possibly empty) connected components. The vertices in one component, the *right part*, have λ-values greater than v, while the vertices in the other component, the *left part*, have λ-values less than v. Let R_0^+ and R_0^- be the regions delimited by the x-axis, D_0, and the vertical lines $x = 1$ and $x = -1$, respectively.

Now we place all seeds and the cover of the right part and the left part in R_0^+ and R_0^-, respectively. Name the children of v in the right part in order of decreasing λ-value $r_1, \ldots, r_t$ and place them from right to left as in the proof of Proposition 8.4 (see Figure 8.13), but with tiny gaps in between the left vertical tangency line of disk D_i and the right vertical tangency line of D_{i+1} for $i = 1, \ldots t-1$. The children $l_1, \ldots, l_t$ of v in the left part are defined and placed symmetrically (with respect to the line $x = 0$). Now we recurse, using the regions defined by each child disk, the vertical lines through its leftmost and its rightmost point, and the x-axis.

(ii) $\Rightarrow$ (iii): Trivial.

(iii) $\Rightarrow$ (ii): We map each vertex v of T to the seed $\lambda(v) \in \mathbb{R}^1$. Let all edges be directed from left to right with respect to the seed ordering on the x-axis and let v_i $(i = 1, \ldots, n)$ be the vertex in T that is mapped to seed $\lambda(v_i) = i$. We insert the edges iteratively by increasing order of the source vertices. For each $i = 1, \ldots, n$ we draw any outgoing edge v_iv_j as the two-segment polyline whose bend is placed at position $(i + 1/2, (j - i)/2^i)$. Figure 8.14 shows an example of this construction. Clearly, all edges with the same source do not intersect (except at their common vertex). It remains to show that none of the previously inserted edges is intersected by a new edge v_iv_j. Since there are no forbidden

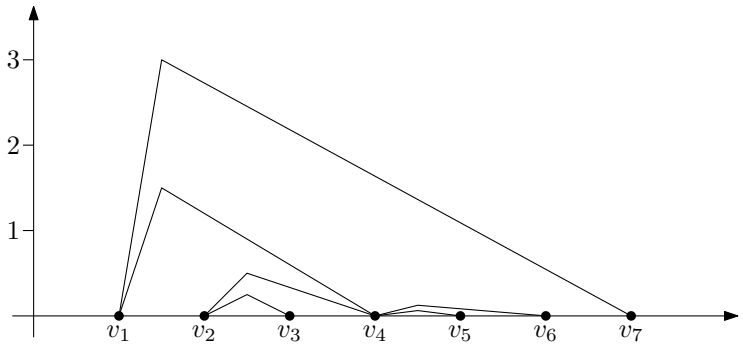


Figure 8.14: Incremental 1d-1BD$^+$ for a tree with no forbidden pairs.

pairs, it holds that for any previously inserted edge v_kv_l and the current edge v_iv_j, the four seeds are ordered as $k < i < j \leq l$. The slope of the right leg of v_kv_l in the drawing is

$$-\frac{1}{2^k} \cdot \frac{l-k}{l-k-1/2} < -\frac{1}{2^k}$$

and the slope of the right leg of the new edge v_iv_j is

$$-\frac{1}{2^i} \cdot \frac{j-i}{j-i-1/2} \geq -\frac{1}{2^{i-1}}.$$

Since $k \leq i-1$ this means that the right leg of any previously inserted edge is steeper than the right leg of v_iv_j. Moreover, the bend of any previously inserted edge is to the left of i. Hence edge v_iv_j does not intersect any other edge when it is inserted. Once all edges are drawn, we have found a valid 1d-1BD$^+$ for T. □

Given the tree, statement (iii) can be checked in $O(n \log n)$ time using an interval tree, therefore we immediately obtain the following corollary.

Corollary 8.1 *Given a λ-labeled tree T, we can decide in $O(n \log n)$ time whether T is λ-realizable as CCG$^+$.*

We now turn to question Q4. So given a set of seeds S, a tree $T = (V, E)$, and a bijection between S and V that assigns each vertex to a seed, we give a decision algorithm for the realizability of T as a triangle-CCG$^+$ on S.

We call a family of homothetic triangles *V-shaped* if each triangle is symmetric to a vertical line and its bottommost vertex is unique. In the following we will consider only V-shaped triangles. First note that there are trees T and seed sets S for which the answer to question Q4 is negative even if the mapping between vertices and seeds is not fixed in advance. Figure 8.15 shows a complete binary tree T on seven vertices and the one-dimensional point set $S = \{a(0), b(2), c(5), d(11), e(13), f(16), g(33)\}$. A case distinction on the seed that represents the root vertex 1 shows that it is not possible to find a representation of T as a triangle-CCG$^+$ on S. The example in Figure 8.15 shows the case where seed g represents the root. In this case any two covers of points in $S \setminus \{g\}$ that touch the covering triangle of g will overlap, even the covering triangles of the most distant seeds a and f. Hence it is impossible to attach the two children to the root.

On the other hand, there is always a tree that can be realized on a given set of seeds as Proposition 8.3 (ii) shows. Below we give an algorithm that decides this realizability

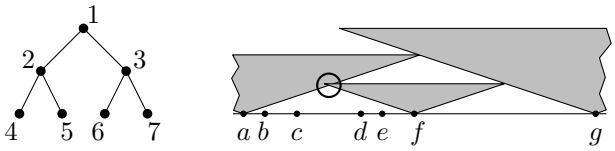


Figure 8.15: Binary tree that is not realizable as triangle-CCG$^+$ on the given seeds.

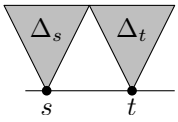


Figure 8.16: Atomic triangles.

question in $O(n \log n)$ time. We say that a set of points on a line is in *general position* if no point is equidistant to two other points, and we consider only such point sets. Furthermore, we define the *atomic* V-shaped covering triangles for a given V-shaped family of triangles a pair of seeds $\{s, t\}$ as the unique congruent triangles Δ_s and Δ_t placed at s and t that touch each other in a triangle vertex, see Figure 8.16.

Algorithm 8.1: V-shaped triangle-CCG$^+$ on fixed seeds realizing a tree.

Input: seed set S with at least two seeds
Output: cover $\mathcal{C}$ of S

```
1  initialize L ← S
2  initialize C ← ∅
3  while |L| > 2 do
4      {s, t} ← closest pair in L
5      {Δ_s, Δ_t} ← pair of atomic triangles for {s, t}
6      choose u ∈ {s, t}
7      add Δ_u to C
8      delete u from L
9  add both atomic triangles for L = {s, t} to C
10 return C
```

Algorithm 8.1 generates a cover that realizes a corresponding tree as triangle-CCG$^+$ on the given seeds. Note that this tree is not unique as the choice of the seed u in line 6 is arbitrary. It is not hard to see that the triangle-CCG$^+$ obtained by Algorithm 8.1 is indeed a tree due to our assumption concerning general position. Now we can finally state our result based on Algorithm 8.1.

Theorem 8.5 *Given a set $S \subset \mathbb{R}^1$ of seeds in general position and a tree T with a fixed seed assignment for each vertex, we can decide in $O(n \log n)$ time whether T can be realized as a V-shaped triangle-CCG$^+$ on S.*

Proof. We construct a cover that realizes T edge by edge. We start with T and an empty cover $\mathcal{C}$. Observe that the closest pair of seeds must form an edge of T, otherwise the triangle-CCG$^+$ of $\mathcal{C}$ would not be connected. Furthermore, one of the vertices of this edge must be a leaf since only one of the atomic triangles can grow any further. Thus, we determine the closest pair of seeds $\{s, t\}$ and check whether st is a leaf edge of T. If

this is not the case, we answer "no". Otherwise we choose the seed u in line 6 as the leaf vertex of edge st. Now Algorithm 8.1 adds the atomic triangle Δ_u for u to $\mathcal{C}$ and removes u from L in line 8. Accordingly, we delete the leaf u and its incident edge in T. By induction the closest pair in L again corresponds to a leaf edge of the modified tree T and we repeat the above process. If the construction terminates without answering "no" in one of the iterations, it is clear that we have constructed a cover $\mathcal{C}$ that represents T as a triangle-CCG$^+$and we answer "yes".

Finding the closest pair of seeds in line 4 is the time-critical part of Algorithm 8.1 and can be done by maintaining a priority queue that is initialized to the distances of all neighboring seeds. This takes $O(n \log n)$ total time. □

From Theorem 8.5 it is clear that Algorithm 8.1 can be used to generate *all* trees that can be realized as triangle-CCG$^+$ on S and thus we obtain the following corollary.

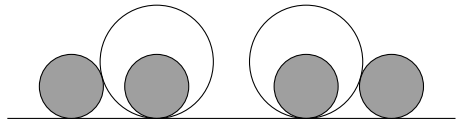
Corollary 8.2 *Let $S \subset \mathbb{R}^1$ be a set of seeds in general position, let $\mathcal{T}(S)$ be the set of trees that are realizable on S as CCG$^+$ with homothetic V-shaped triangles as covering objects, and let $\mathcal{M}(S)$ be the set of trees that can be obtained by Algorithm 8.1. Then $\mathcal{T}(S) = \mathcal{M}(S)$.*

Proof. As all triangle-CCG$^+$'s obtained by Algorithm 8.1 are trees that are realizable on S it remains only to show that $\mathcal{T}(S) \subseteq \mathcal{M}(S)$. So consider any tree $T \in \mathcal{T}(S)$ and its realization as triangle-CCG$^+$. Since covering objects are V-shaped homothetic triangles the y-coordinate of the contact point between two triangles is proportional to the distance of the covered seeds. We sort the triangles in the realization of T in increasing order by their height and impose the same order on the associated seeds. Then we apply Algorithm 8.1 under the constraint that u (and the triangle Δ_u) in line 6 is chosen according to this ordering of the seeds. It follows that in each iteration the point covered by the lowest remaining triangle in the given triangle-CCG$^+$ of T is a point of the closest pair. Hence the algorithm is able to reconstruct T. □

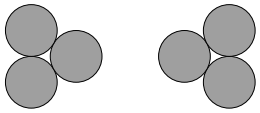
Note that the restriction to V-shaped triangles in Theorem 8.5 and Corollary 8.2 is made only for ease of presentation. The results for V-shaped triangle families can easily be extended to families of homothetic triangles whose top sides are parallel to the x-axis. It suffices to redefine the atomic triangles as the unique congruent triangles in the given family that touch each other in a triangle vertex.

8.5 Disk or Triangle Seeds in the Plane

In this section, we consider a different class of seeds, namely disks or homothetic triangles in the plane. We will cover them with the same kind of objects, that is, the covers for disks are disks and the covers for homothetic triangles are homothetic triangles. If the seeds are not points the main difference is that the minimal size of each cover is bounded, so the results differ in many cases from those obtained in the previous sections when the seeds were points. We first consider seeds in $\mathbb{R}^2_+$ that are tangent to the x-axis (for triangles the bottom vertex is on the x-axis) and then how to translate the results to general seeds in the plane.



(a) Disks on the x-axis without connected CCG$^+$.



(b) Disks in the plane without connected CCG.

Figure 8.17: Disk seeds that cannot be covered by a connected disk cover. Seeds are drawn in gray, covers in white.

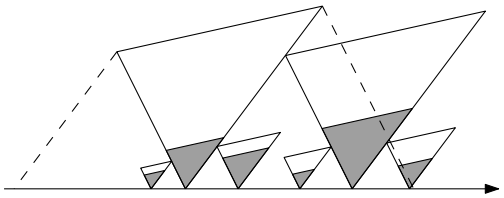


Figure 8.18: There is always a connected CCG$^+$ for homothetic triangle seeds on a line.

8.5.1 Connectivity

Unlike the connectivity results for points we can neither guarantee the existence of a connected CCG$^+$ for disk seeds tangent to a horizontal line nor the existence of a connected CCG for disk seeds in the plane, see Figure 8.17. For homothetic triangles, however, we can find a connected CCG$^+$ as the next proposition shows.

Proposition 8.5 *For a set S of homothetic-triangle seeds on a line there is always a connected triangle-CCG$^+$.*

Proof. We consider the family of parallel lines induced by the triangle sides opposite to each bottom vertex of the seeds. Among these lines there is one line g that contains all seed triangles in its lower half space. We cover the seed belonging to g by a big triangle such that the interval on the x-axis between the projections of the two vertices of the top edge along the direction of the respective opposite triangle sides contains all bottom vertices of the seeds, see the dashed projection lines in Figure 8.18. Then we inflate a covering triangle for each of the seeds until it touches one of the previous triangles. Due to the size of the first covering triangle each inflated triangle eventually touches another covering triangle and hence the CCG$^+$ is connected. □

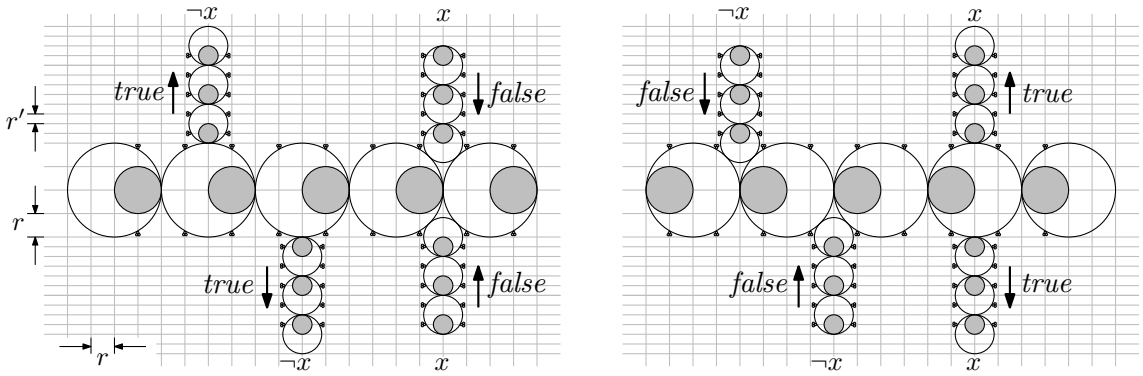
The same method can be extended to find a connected CCG for any set of general homothetic-triangle seeds in the plane.

Corollary 8.3 *For a set S of homothetic-triangle seeds there is always a connected CCG.*

Proof. The initial seed to be covered is selected as before and the size of its covering triangle must be large enough such that all other seeds are contained in the (unbounded) region between the two projection lines. This guarantees that inflating covering triangles for the other seeds will always lead to a contact event. □

We cannot extend the result of Proposition 8.5 any further. It is easy to see that in the example of Figure 8.18 no biconnected CCG$^+$ exists (for example, the covering triangle of the rightmost seed can never touch any covering object other than its left neighbor). For disk seeds deciding whether there is a connected CCG turns out to be hard.

Figure 8.19: Stopper element.



(a) Variable x is *false*. (b) Variable x is *true*.

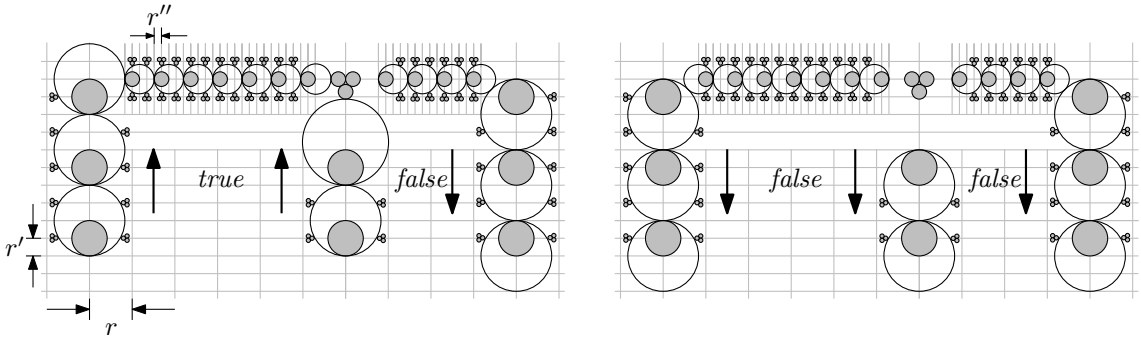
Figure 8.20: The variable gadget.

Theorem 8.6 *Given a set S of disk seeds, it is NP-hard to decide whether there is a connected CCG on S, even if there are only four different seed radii.*

Proof. The proof is by reduction from NP-complete problem Planar3-Sat [Lic82] that has been introduced in Section 2.3. So given a planar Boolean 3-CNF formula φ, We construct two types of gadgets, one for the variables of φ and one for the clauses of φ. First, as an important building block, we define a stopper element consisting of three congruent and pairwise touching disks as depicted in Figure 8.19. Observe that these disks can be covered only by themselves—any larger cover of any disk would intersect the others.

Variable gadgets. For each variable of φ we place a set of congruent disk seeds with radius r horizontally on a grid as shown in Figure 8.20. Both below and above the disks we place small stopper elements such that, in order to touch them all in a CCG, each of the variable seeds must be covered by a disk of radius $2r$. These covers must be centered on the same horizontal line as the seeds and they are either tangent to the leftmost point or to the rightmost point of their respective seed. We define the cover in Figure 8.20a as encoding the value *false* and the alternative cover in Figure 8.20b as encoding the value *true*. In order to transmit the truth value into the clauses we connect vertical literal "wires" to the variable gadget. The wires consist of seeds of radius $r' = (\sqrt{2}-1)r$ flanked by stopper elements as before. For negated (non-negated) literals the wires are centered on the vertical grid line tangent to the *left* (*right*) of a seed of the variable gadget. Hence, in order to form a connected CCG, the covers of literal wires that are *false* are pulled towards the variable and the covers of literals that are *true* are pushed away from it as illustrated in Figure 8.20.

Clause gadgets. The clause gadget is depicted in Figure 8.21 and combines three literal wires in a comb-like shape with a stopper element in the center. The left and right wires make a 90-degree turn similar to the connection of a literal wire to its variable gadget. Hence the horizontal literal wires use seeds of radius $r'' = (\sqrt{2}-1)r'$. We use the same radius r'' for the disks of the central stopper element. The wires stop in front of the center



(a) Two literals are *true* and connect to the central stopper element.

(b) All literals are *false* and the central stopper element cannot be connected.

Figure 8.21: The clause gadget.

of the clause, more precisely, they stop with the last seed of the wire that can be placed without intersecting the central stopper element. The very last seeds of the three wires are not framed by small stopper elements any more. Observe that in this construction literal wires that are pushed into the clause (transmitting the value *true*) are able to connect to the central stopper element by inflating the final covering disk appropriately (see Figure 8.21a), while wires that are pulled away from the clause (transmitting the value *false*) cannot connect to it. Inflating the final covering disk while keeping the contact to the previous disk in the wire would lead to an overlap with the last stopper elements in the wire (see Figure 8.21b). Hence the central stopper element of a clause gadget can be connected to the remaining graph if and only if one of the literal wires transmits the value *true*.

Reduction. In order to obtain a connected CCG, it is necessary to connect the stopper element in the center of each clause to at least one literal wire. By construction this is possible if and only if there is a variable assignment that satisfies all clauses. We further need to ensure that all the variable gadgets, which are placed on a horizontal line, are in the same connected component of the CCG. This can easily be achieved by placing a single disk seed between any two neighboring variable gadgets that can be covered by a disk that touches both neighboring gadgets irrespective of their truth values. Now it is easy to see that all variable gadgets and all literal wires are in a single connected component. Hence the whole CCG is connected if and only if each clause gadget is also connected to that component by at least one literal wire and that is the case if and only if each clause is satisfied by the truth value assignment encoded in the variable gadgets.

The variable-clause graph H_φ of φ can be embedded on a grid whose size is quadratic in the size of φ. Hence the grid on which the disk seeds in our construction are placed also has polynomial size and the reduction takes polynomial time. We have used only four seed radii. □

8.6 Concluding Remarks

In this chapter we have studied cover contact graphs, a new class of graph representations, which are contact graphs of two-dimensional geometric objects (disks or triangles) that are

pinned down to some degree by the requirement to cover a given set of seeds (points, disks, or triangles) in the plane. Hence cover contact graphs are found in the cross section of general contact graph representations and geometric covering problems. We have examined two questions, namely whether the given seeds admit a *connected* cover contact graph and whether a given graph can be *realized* on the given seeds as a cover contact graph. We have shown that the connectivity question can be answered positively for point and triangle seeds but it is NP-hard for disk seeds. The realization question is NP-hard already for point seeds, but we have given two decision algorithms for trees that are to be realized on point seeds on a line.

Open problems. Since cover contact graphs are a new topic in the area of graph representations many interesting questions remain open. We know that every 3-vertex graph can be represented as CCG on any set of three points. We have further given an example of six points whose Delaunay triangulation is not representable as a CCG. The status of this question for four and five points and planar graphs with four and five vertices is still unknown. If we consider a set of point seeds in convex position, is there always a triangulation that can be realized as CCG?

We have positively answered the connectivity question for point seeds, even a 2-connected CCG is always possible. Does this extend to 3-connectivity, that is, does any set of points admit a 3-connected CCG? For disk seeds we have shown that it is NP-hard to decide whether a connected CCG exists. Is this decision problem still NP-hard if we ask for a connected CCG$^+$ for disk seeds in $\mathbb{R}^2_+$ that touch the x-axis?

In Theorem 8.4 we have characterized vertex-labeled trees that can be realized as disk-CCG$^+$ on a set points on a line in a prescribed vertex order. Can we give an equivalent characterization of vertex-labeled trees that can be realized as disk-CCG, where the difference to the given characterization is that disks may now touch on both sides of the x-axis.

Finally, all our questions can be extended to other classes of seed objects and covering objects.

Chapter 9

Consistent Digital Rays

In this chapter we are considering a very fundamental and classic problem in digital geometry, namely how to best represent a line segment in a raster graphics. This topic has a direct impact to the visualization of geometric networks, where each edge is represented as the line segment between the points representing its incident vertices. In Euclidean geometry the line segment between two points is a well-defined object. And it is usually in terms of Euclidean geometry that we think about visualizations of networks. However, if a geometric network is to be displayed on a grid-based medium like a computer screen it is essential to have a mapping from the Euclidean space to the display medium that is consistent in a certain sense.

We show that, although being a well-studied subject, it is not obvious how to consistently define line segments in a grid topology and at the same time maintain a certain visual similarity between the Euclidean line segment and the grid line segment. We propose a set of consistency axioms for line segments in the grid and consider the subproblem of representing segments of rays from a fixed origin o in the grid. We recursively define a system of consistent rays in the grid that asymptotically optimally represent the corresponding Euclidean rays in terms of the Hausdorff distance. This chapter is based on joint work with Jinhee Chun, Matias Korman, and Takeshi Tokuyama [CKNT08, CKNT09].

9.1 Introduction

The *digital line segment* $\mathrm{dig}(pq)$ between two grid points p and q is a fundamental digital geometric object, but still its definition is not that obvious. Indeed, the digital representation of line segments has been an active area of research for almost half a century now (see, for example, the survey of Klette and Rosenfeld [KR04]). In digital geometry, a geometric object is represented by a set of d-dimensional grid points in a digital grid $\mathbf{G} = \mathbb{Z}^d$, and its topological properties are considered under a grid topology defined by a graph on the grid. In two dimensions, it is common to consider the *orthogonal* (or *4-neighbor*) grid topology, where each point $p = (x, y)$ is connected to its four vertical and horizontal neighbors $(x, y-1), (x, y+1), (x-1, y)$, and $(x+1, y)$; we focus on this topology. However, as a variant, we may consider the *8-neighbor* grid topology that connects each grid point $p = (x, y)$ to its 4-neighbors as before and additionally to its diagonal neighbors $(x+1, y-1), (x+1, y+1), (x-1, y-1)$, and $(x-1, y+1)$. Given a grid topology, the digital line segment $\mathrm{dig}(pq)$ between two grid points p and q is a path between p and q in this topology.

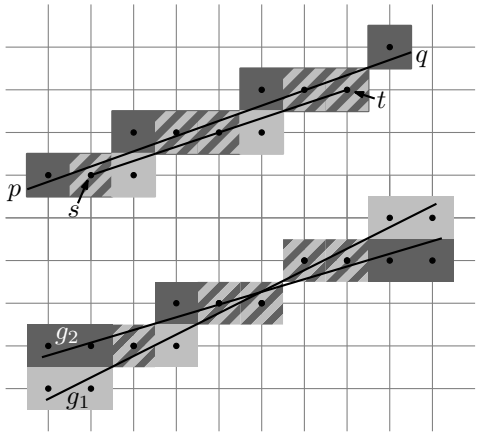


Figure 9.1: Euclidean line segments and their DSSs. Intersections are indicated by bicolored pixels. Axiom (S3) is violated since $s, t \in \mathrm{dig}(pq)$ but $\mathrm{dig}(st) \not\subseteq \mathrm{dig}(pq)$ (top); the intersection of the DSSs g_1 and g_2 is not connected (bottom).

Since a digital line segment $\mathrm{dig}(pq)$ is a representation of a line segment $\overline{pq}$ in Euclidean geometry, it is natural (at least from a mathematical perspective) to set up the following axioms that a digital line segment should satisfy:

(S1) A digital line segment $\mathrm{dig}(pq)$ is a connected path between p and q under the grid topology.
(S2) For any two grid points p and q, there is a unique digital line segment $\mathrm{dig}(pq) = \mathrm{dig}(qp)$.
(S3) For a digital line segment $\mathrm{dig}(pq)$ and two grid points $s, t \in \mathrm{dig}(pq)$, it holds that $\mathrm{dig}(st) \subseteq \mathrm{dig}(pq)$.
(S4) For any two grid points p and q there is a grid point $r \notin \mathrm{dig}(pq)$ such that $\mathrm{dig}(pq) \subset \mathrm{dig}(pr)$.

Note that axiom (S3) implies that a non-empty intersection of two digital line segments is either a grid point or a digital line segment. Axiom (S4) implies that any digital line segment can be extended to a digital line. We often identify a path in a grid with its vertex set if the correspondence is clear. Accordingly, if we say that a grid point p is in a path P, it means that p is a vertex of P.

Unfortunately, popular definitions of two-dimensional digital line segments in computer vision do not satisfy these axioms. For example, in the standard definition of a *digital straight segment (DSS)* [KR04], a digital line segment (in the 8-neighbor topology) that corresponds to the Euclidean line segment given by $y = mx + b$, $x_0 \leq x \leq x_1$ is defined as the set of grid points $\{(i, \lfloor mi + b + 0.5 \rfloor) \mid x_0 \leq i \leq x_1\}$ for $|m| \leq 1$. Using this definition the intersection of two DSSs is not always connected, and axiom (S3) is violated in some cases as depicted in Figure 9.1.

In the two-dimensional grid, another possibility to define digital line segments would be to use the system of L- and Γ-shaped shortest paths. An L- or Γ-shaped path between two points $p = (x_p, y_p)$ and $q = (x_q, y_q)$ such that $x_p \leq x_q$, is the (at most) 2-link path that consists of the grid points on the vertical segment pp' and on the horizontal segment $p'q$ where $p' = (x_p, y_q)$. It is easy to confirm that the system of these paths satisfies axioms (S1)–(S4) for digital line segments. A clear drawback is that an L-shaped path is visually very different from the Euclidean line segment, and the Hausdorff distance from $\overline{pq}$ to the

L-shaped path between p and q becomes $n/\sqrt{2}$ for $p = (0, n)$ and $q = (n, 0)$. If, on the other hand, one accepts to use a non-planar graph structure to define the topology on the grid points, Pach et al. [PPS90] show that the shortest-path distance (using Euclidean distance for the edge lengths) in the grid topology given by a suitable sparse graph is at most $(1 + \epsilon)$ times the Euclidean distance. Accordingly, the polygonal path consisting of the edge set of the shortest path between p and q in the graph gives a nice approximation of the line segment $\overline{pq}$. However, the graph structure is a union of many randomly chosen lattice structures on the grid points using long edges with a variety of slopes; thus, the vertex set of the polygonal path is too sparse for direct use as a digital line segment. Also, the method does not guarantee an $o(n)$ bound for the Hausdorff distance.

Thus, it seems that there is a trade-off between the axiomatic requirements and the visual quality of digital line segments. It is a challenging problem to find a system of digital line segments that satisfies the axioms and is visually alike Euclidean line segments at the same time.

Here, we study a less ambitious but important subproblem, motivated by geometric optimization applications: we consider only digital line segments that have the origin o as one of their endpoints. In other words, we consider digital halflines emanating from o. Then $\mathrm{dig}(op)$ is defined as the unique portion of the halfline that has p as its second endpoint. We call such segments *digital ray segments* or simply *digital rays* emanating from o. For digital rays, axioms (S1)–(S4) for digital line segments are adapted as follows:

(R1) A digital ray $\mathrm{dig}(op)$ is a connected path between o and p under the grid topology.
(R2) There is a unique digital ray $\mathrm{dig}(op)$ between o and any grid point p.
(R3) For a digital ray $\mathrm{dig}(op)$ and a grid point $r \in \mathrm{dig}(op)$, it holds that $\mathrm{dig}(or) \subseteq \mathrm{dig}(op)$.
(R4) For any grid point p, there is a grid point $r \notin \mathrm{dig}(op)$ such that $\mathrm{dig}(op) \subset \mathrm{dig}(or)$.

We also give one additional *monotonicity* axiom, which is not combinatorial, but a reasonable condition for a digital ray:

(R5) For any $r \in \mathrm{dig}(op)$, $|\overline{or}| \leq |\overline{op}|$, where $|\overline{ab}|$ is the length of the Euclidean segment $\overline{ab}$.

A system of digital rays is called *consistent* if it satisfies axioms (R1)–(R5). From these axioms it follows that the union of all digital rays forms an infinite spanning tree T of the grid graph on $\mathbf{G}$ rooted at o, such that $\mathrm{dig}(op)$ is the unique path between o and p in the tree. Because of axiom (R4), T cannot have leaves. Thus, the problem is basically to embed the infinite "star" consisting of the halflines emanating from o in the d-dimensional Euclidean space as a tree in the d-dimensional grid. Although embedding a tree in a grid is a popular topic in metric embedding and graph drawing, it is a novel and interesting problem to geometrically approximate ray segments by paths.

Motivation. At first sight, studying consistent digital rays and their properties may seem like a fairly restricted problem; yet, a consistent system of digital rays enables us to define star-shaped regions in the grid $\mathbf{G}$ that are centered at some grid point o. These digital analogues of star-shaped regions are used in optimization problems in a pixel grid. A square *pixel grid* is a subdivision of an $n \times n$ square region into n^2 unit squares called *pixels*. We have a canonical one-to-one correspondence between pixels in a pixel grid $\mathbf{P}$ and grid points in the two-dimensional grid $\mathbf{G}$ restricted to an $n \times n$ subgrid. Thus, we can translate the definitions of digital rays and digital star-shaped regions in $\mathbf{G}$ to their counterparts in $\mathbf{P}$. A *pixel grid image* is an assignment of a color to each pixel. A monochromatic image can be considered as a function from the set $\mathbf{P}$ of all pixels to real values in $[0, 1]$ called

gray levels, while a color image, for example taken with a digital camera, can be considered as a triple of functions from $\mathbf{P}$ to real values in $[0, 1]$ corresponding to the color levels of red, green, and blue.

Image segmentation is an important task in computer vision, which separates an object from the background in the picture. Asano et al. [ACKT01] formulated the problem as a least-squares optimization problem and gave an efficient algorithm if the object is a region bounded by two x-monotone curves. Several improved results such as controlling smoothness of curves and higher dimensional extensions were given by Wu and Chen [WC02], and the optimal-ratio formulation was given by Wu [Wu06]. Wu further pointed out that image segmentation problems appear in medical applications. For example, some tumors can be approximated by layers of concentric three-dimensional star-shaped annuli, where a star-shaped annulus is the set difference of two star-shaped regions with a common center o. Similarly, layers of plaques in diseased arteries form concentric star-shaped annuli in the lumina of arteries. Certain medical imaging methods, for example optical coherence tomography with an intravascular probe, transform concentric annuli into regions between x-monotone curves. Wu [Wu06] applied his algorithm to extract regions between x-monotone curves to such medical image data. For images that have not been projected in a way that transforms concentric annuli into regions between x-monotone curves, for example, X-ray computed tomography images, the methods of Wu cannot be applied directly to extract the regions of interest. However, we can define a mapping of $\mathbf{P}$ based on a system of consistent rays that transforms the image as required, apply Wu's algorithm, and then use the inverse mapping to find the corresponding annular region in the original image.

Chen et al. [CCKT04] and Chun et al. [CST03] considered the *pyramid approximation* problem to compute the least-squares approximation of an input terrain (given as a function on $\mathbf{P}$) where each horizontal slice (that is, a region bounded by a contour line) of the output terrain is a special kind of rectilinear convex region as shown in Figure 9.2, where heights are given by gray-levels. It was left as an open problem to solve the analogous *mountain approximation* problem where each horizontal slice is a star shape. In Section 9.4 we show how our definition of consistent digital rays can be used to solve the mountain approximation problem.

Related problems. In computational geometry, the problem of representing geometric objects in digital geometry without causing topological and combinatorial inconsistencies is a major concern, and algorithmic solutions have been considered from the viewpoint of robust, finite-precision geometric computation [GY86, Sug01].

Suppose that we want to represent a set S of line segments digitally in a pixel grid. Although ideally one would like to give a precisely defined and consistent system of digital line segments, the difficulties mentioned in the previous section prevent us from doing so. Rather, it is popular to use a *dynamic* method to digitize the line segments; that is, the digital approximation of a line segment ℓ is affected by the configuration of the other line segments of S. In particular, it is required to construct the arrangement of S in the digital plane without changing the combinatorial structure of the arrangement. At the same time all vertices of the arrangement must be located at grid points, and each line segment shall be visually alike the original line segment. It is known that a grid of exponential size is necessary to represent all the combinatorial types of arrangements of n straight line segments [GPS89]; hence lines need to bend if we want to use a polynomial-size grid.

In the pioneering paper of Greene and Yao [GY86] and the follow-up work by Goodrich et al. [GGHT97], each line segment is represented by a polygonal chain consisting of edges

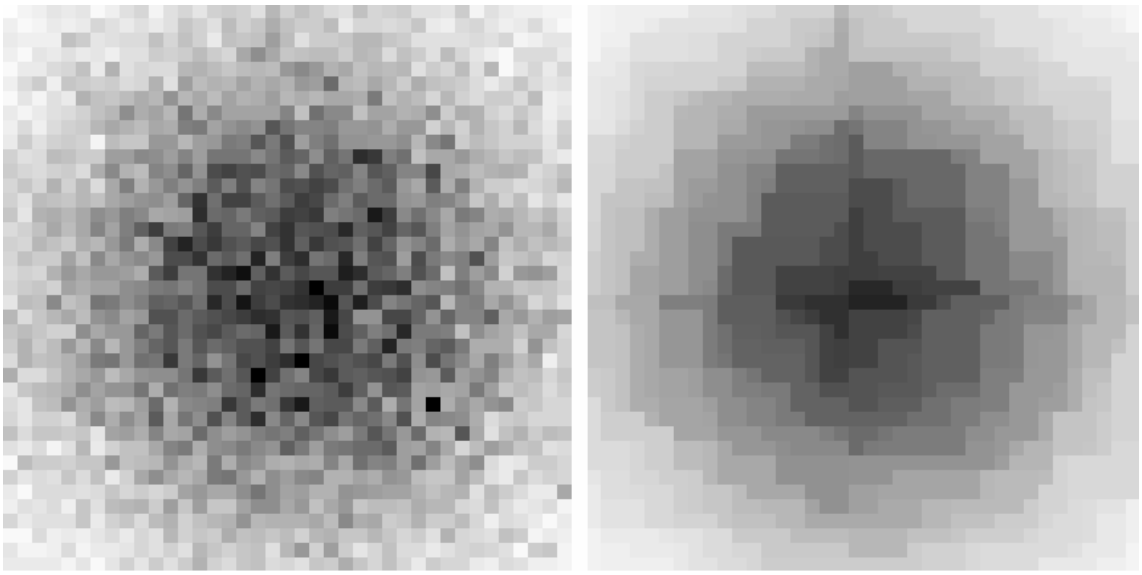


Figure 9.2: Input terrain (left) and pyramid approximation (right). Height values are depicted as gray levels.

of the arrangement. It is necessary to carefully round each vertex of the arrangement to a grid point in order to avoid combinatorial inconsistencies, and a method named *snap rounding* was proposed. Since no two edges of the arrangement intersect each other, we can draw edges using a popular method like DSS once we have a suitable representation of the arrangement. We note that the snap rounding idea is important not only in the theory of robust computation but also in the practical design of geometric editors or systems: for example, the Ipe drawing editor [Sch95] is a pioneering system that uses snap rounding; the idea has also been implemented in the CGAL project [cga].

This dynamic approach is different from our static approach, in which each digital line segment is defined irrespective to the existence of other lines in the arrangement. In spite of the success of the dynamic approach, we think that it is important to investigate how well line segments can be digitized statically and to consider the combination of static and dynamic methods to design efficient systems and algorithms in digital geometry.

Contributions. Our main result in this chapter is an asymptotically tight $\Theta(\log n)$ bound for the Hausdorff distance between $\mathrm{dig}(op)$ and $\overline{op}$ maximized over all points p in an $n \times n$ grid (Section 9.2). The lower bound argument is based on discrepancy theory, and the upper bound is attained by a systematic recursive construction of a two-dimensional spanning tree of the $n \times n$ grid. Surprisingly, if we do not include the monotonicity axiom (R5), we can reduce the bound to $O(1)$. In Section 9.3, the construction is extended to the d-dimensional case. Finally, in Section 9.4, we present how our system of consistent digital rays can be used to define digital star shapes, which in turn are useful for the mountain approximation problem.

9.2 Digital Rays in the Two-Dimensional Grid

In this section we derive the lower and upper bound for the Hausdorff distance between a Euclidean line segment and its digital counterpart in two dimensions. Section 9.3 will discuss the extension of the results to the d-dimensional case.

We start with some basic notions. The Hausdorff distance $H(A, B)$ of two objects A and B is defined as $H(A, B) = \max\{h(A, B), h(B, A)\}$, where $h(A, B) = \max_{a \in A} \min_{b \in B} d(a, b)$ and $d(a, b)$ is some distance between the points a and b. Although it is most natural to consider the Euclidean distance for $d(a, b)$, we will use the L^∞-metric in the following for technical convenience. Since the ratio of the Euclidean distance to the L^∞-distance in d-dimensional space is in the interval $[1, \sqrt{d}]$, the choice of the metric is irrelevant in a constant-dimensional space when considering bounds in big-O and big-Ω notation.

Consider the two-dimensional integer grid $\mathbf{G} = \{(i, j) \mid i, j \in \mathbb{Z}\}$, which we may also interpret as a vertex set $V = \mathbf{G}$. We then define a planar graph G on V that represents the adjacency relations of a 4-neighborhood pixel grid. In $G = (V, E)$ each vertex (i, j) is connected to its four neighbors $(i, j-1)$, $(i-1, j)$, $(i+1, j)$, and $(i, j+1)$. This graph also defines the orthogonal topology of the grid $\mathbf{G}$. A subset of V (and thus of $\mathbf{G}$) is *connected* in this topology if its induced subgraph in G is connected.

9.2.1 The Lower Bound Result

We focus on the part $\mathbf{G}(n)$ of the planar orthogonal grid restricted to the region defined by $x + y \leq n$ in the first quadrant. The remaining quadrants are handled analogously by rotating $\mathbf{G}(n)$ around the origin. From the monotonicity axiom (R5) it follows that $\mathrm{dig}(op) \subset \mathbf{G}(n)$ for any $p \in \mathbf{G}(n)$ and that $\mathrm{dig}(op)$ is a shortest path in the grid. We show that there exists a point $p \in \mathbf{G}(n)$ such that the Hausdorff distance $H(\mathrm{dig}(op), \overline{op})$ is $\Omega(\log n)$.

To derive the lower bound we use a classical result on pseudo-random number generation [Mat99, Nie92, Sch77]. The following historical summary is according to Schmidt's textbook [Sch77]. Consider a sequence $X = (x_0, x_1, x_2, \ldots)$ of real numbers in $[0, 1]$. For any given $a \in [0, 1]$ and $m \in \mathbb{N}$, define $X_m(a) = |\{0 \leq i \leq m \mid x_i \in [0, a]\}|$. The *discrepancy* of the sequence $X_m = (x_0, x_1, \ldots, x_m)$ is defined as $\sup_{a \in [0,1]} |am - X_m(a)|$. Van der Corput conjectured in 1935 that for any sequence X, the discrepancy cannot be bounded by a constant (indeed, 1, in the original conjecture). This was answered affirmatively by van Aardenne-Ehrenfest in 1945. Roth gave an $\Omega(\sqrt{\log n})$ bound in 1954, and the correct order of magnitude of the discrepancy is $\Theta(\log n)$ as shown by Schmidt in 1972. We make use of discrepancy theory in the form of Theorem 9.1 below. We remark that a slightly stronger version of van der Corput's conjecture was given in a list of favorite questions of Erdős [Erd64]: He conjectured that there is a real number a such that $\max_{m<n} |am - X_m(a)|$ is an unbounded function in n. In fact, Schmidt's method yields a $\Theta(\log n)$ bound for this function.

Theorem 9.1 (Schmidt [Sch72]) *Given a sequence $X = (x_0, x_1, x_2, \ldots)$ of real numbers in $[0, 1]$ and a sufficiently large integer n, there is an integer $m < n$ and a real number $a \in [0, 1]$ such that the subsequence $X_m = (x_0, x_1, \ldots, x_m)$ satisfies $|am - X_m(a)| > c \log n$, where c is a positive constant independent of n.*

Let T be the spanning tree of $\mathbf{G}(n)$ that is the union of $\mathrm{dig}(op)$ for all $p \in \mathbf{G}(n)$. Figure 9.3 shows the spanning tree induced by a set of consistent rays. We will apply

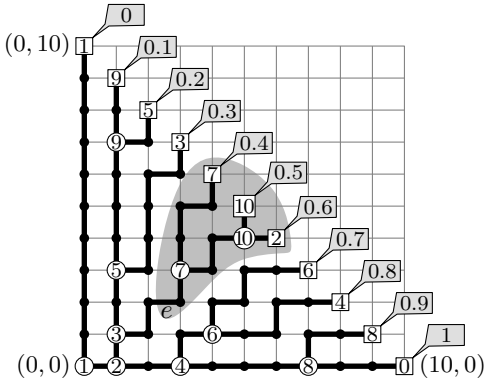


Figure 9.3: A spanning tree T of $\mathbf{G}(n)$ for $n = 10$. The labels attached to the leaves as callouts are the elements of the low-discrepancy sequence $X(T)$ in the interval $[0, 1]$, while the inner labels are used in the construction of that sequence and determine the order of the elements.

Theorem 9.1 to a sequence constructed from T to obtain a lower bound for the Hausdorff distance between $\mathrm{dig}(op)$ and $\overline{op}$. Let's start with some preparations to construct that sequence.

For $m = 1, 2, \ldots n + 1$, let $L(m) = \{(i, m - 1 - i) \mid i = 0, \ldots, m - 1\}$ be the subset of $\mathbf{G}(n)$ satisfying $x + y = m - 1$.

Lemma 9.1 *For any integer m, $1 \leq m \leq n$, the spanning tree T has a unique node of degree 3 in $L(m)$.*

Proof. There are m grid points in $L(m)$ and $m + 1$ grid points in $L(m + 1)$. Since T does not have any leaves in $L(m)$ for $m \leq n$, the m points of $L(m)$ must all be connected to a point in $L(m + 1)$ and, conversely, the $m + 1$ points of $L(m + 1)$ must all be connected to a point in $L(m)$. In the 4-neighbor grid topology, this is only possible if there are exactly one node of degree 3 and $m - 1$ nodes of degree 2 in $L(m)$. □

We denote this unique degree-3 node in $L(m)$ as the *branching node* of $L(m)$. There is a vertical and a horizontal edge incident to the branching node that lead to its two children in $L(m + 1)$. We denote these two edges as the *branching edges* of $L(m)$.

We associate the number j/n to the leaf $(j, n - j) \in L(n + 1)$ in order to obtain the set $N = \{j/n : j = 0, 1, 2, \ldots n\} \subset [0, 1]$ of leaf labels, see Figure 9.3. For an edge $e = uv$ in T, where u is the parent of v, we define the *subtree rooted at* e to be the subtree of T rooted at the child node v of e. Then for each edge e of T in $\mathbf{G}(n)$, the set of leaves of $L(n + 1)$ in the subtree rooted at e are consecutive, and their associated numbers form an interval $I(e) \subset N$. Let $x(e)$ denote the largest element in $I(e)$. An example for an edge e is given in Figure 9.3, where $I(e) = \{0.4, 0.5, 0.6\}$ and $x(e) = 0.6$.

We create a sequence $X(T) \subset [0, 1]$ as follows: we set $x_0 = 1$, and for $m = 1, \ldots, n$, we set $x_m = x(e_m)$, where e_m is the upper (vertical) branching edge in $L(m)$. Note that for any two different vertical branching edges e and e' the numbers $x(e)$ and $x(e')$ differ since the path from e to the leaf with the largest associated value in $I(e)$ always uses the horizontal branching edge at each encountered branching node. Thus, the obtained sequence $X(T) = (x_0, x_1, \ldots, x_n)$, is a permutation of N that depends only on T. For example, the tree T in Figure 9.3 creates the sequence $X(T) = (1, 0, 0.6, 0.3, 0.8, 0.2, 0.7, 0.4, 0.9, 0.1, 0.5)$. The labels inside the nodes in Figure 9.3 show the correspondence between the unique internal

branching node in $L(i)$ and the leaf located at $(nx_i, n - nx_i)$ in $L(n+1)$ that is associated with the number x_i shown in the callouts. For each $i = 1, \ldots, n$, the corresponding nodes are labeled by i. In other words, each branching node and the rightmost leaf in the subtree rooted at the upper branching edge of that node have the same label.

Let $E(m)$ be the set of edges in T going from $L(m)$ towards $L(m+1)$.

Lemma 9.2 *Let e and f be edges in $E(m)$. If e is to the left of f (that is, the endpoint of e in $L(m+1)$ has smaller x-coordinate than the endpoint of f in $L(m+1)$), we have $x(e) < x(f)$.*

Proof. Assume to the contrary that $x(e) \geq x(f)$. The case $x(e) = x(f)$ contradicts the fact that T is a tree since we would have two different paths from the root to the same leaf. In the case $x(e) > x(f)$, the paths from e and f to their largest leaves must cross. But since the grid topology allows only horizontal and vertical edges between adjacent grid points, the paths must cross in a common tree node, which again contradicts the fact that T is a tree. □

Lemma 9.3 *The set $\{x(e) : e \in E(m)\}$ equals the set $\{x_0, x_1, x_2, \ldots, x_m\}$.*

Proof. A simple induction shows the lemma. First, let $m = 1$, and thus in all valid spanning trees, $E(m)$ consists of the vertical and horizontal edge leaving the origin. We denote these edges by e_v and e_h, respectively. By definition we have $x_1 = x(e_v)$. For e_h, $x(e_h) = 1$ is the largest element (attached to the leaf $(n, 0)$) in $I(e_h)$. Hence, we indeed have $x(e_h) = x_0$.

Now assume that the statement holds for some m. All edges in the set $E(m+1)$, except the two branching edges, just continue the corresponding predecessor edge in $E(m)$ and thus have the same x-values as their predecessor edges. It remains to consider the branching node u in $L(m+1)$. Let e be the edge incident to u in $E(m)$, and let e_v and e_h be the vertical and the horizontal branching edge incident to u in $E(m+1)$. By definition $x_{m+1} = x(e_v)$; furthermore, we have $x(e) = x(e_h)$ since $I(e_h) \subset I(e)$ contains (by Lemma 9.2) the largest element $x(e)$ of $I(e)$. Thus, the statement also holds for $m+1$.□

The following theorem shows our lower bound.

Theorem 9.2 *For any spanning tree T, there is a grid point $p \in L(n+1)$ and a grid point $q \in \mathbf{G}(n)$ such that q is on the path $\mathrm{dig}(op)$ in T and the L^∞-distance from q to the line segment $\overline{op}$ exceeds $c \log n - 1$, where c is the constant considered in Theorem 9.1.*

Proof. To prove the theorem we consider the discrepancy of the sequence $X(T)$. From Theorem 9.1 we have a real number $0 \leq a \leq 1$ and two integers $m < n$ for n large enough such that $|am - X_m(a)| > c \log n$. In the following there are two cases to consider for this inequality, in each of which we determine a grid point q based on the value of $X_m(a)$. This point q is part of a digital ray $\mathrm{dig}(op)$ for some point $p \in L(n+1)$. We use the discrepancy of $X(T)$ to show a lower bound on the distance between $\overline{op}$ and q.

Case 1: $X_m(a) > am + c \log n$. Consider the node q in $L(m+1)$ located at the grid point $(X_m(a) - 1, m - (X_m(a) - 1))$, and let e be the edge between q and its parent in T. By definition, q is on the path $\mathrm{dig}(op)$ from o to the node $p = (x(e)n, n - x(e)n) \in L(n+1)$. Because of the definition of $X_m(a)$ and Lemma 9.3, the set $\{f \in E(m) \mid x(f) \leq a\}$ has cardinality $X_m(a)$. However, there are also exactly $X_m(a)$ edges of $E(m)$ to the left of e, including e itself, since q is the $X_m(a)$-th node in $L(m+1)$ counted from the left. Lemma 9.2 implies that no edge g to the right of e can attain $x(g) \leq a$. Thus, e itself

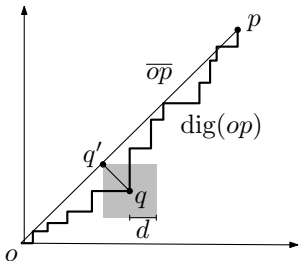


Figure 9.4: The L^∞-distance between the point q on dig(op) and the Euclidean line segment $\overline{op}$ is d. The gray square indicates the set of all points with L^∞-distance at most d from q and point q' is the L^∞-nearest point on $\overline{op}$.

must satisfy $x(e) \leq a$. Now, consider the L^∞-distance of the line segment $\overline{op}$ and q. The line segment $\overline{op}$ is defined by $y = (1/x(e) - 1) \cdot x$ and the L^∞-nearest point from q on $\overline{op}$ is the intersection point q' of $\overline{op}$ and the line $y = -x + m$ through q with slope -1. The intersection point q' has coordinates $(x(e)m, m - x(e)m)$ and its L^∞-distance from q is $(X_m(a) - 1 - x(e)m) \geq (X_m(a) - 1 - am) > c \log n - 1$. Figure 9.4 sketches the situation.

Case 2: $X_m(a) < am - c \log n$. Consider the node q in $L(m+1)$ located at the grid point $(X_m(a), m - X_m(a))$ and the edge e between q and its parent. Since there are only $X_m(a)$ edges $f \in E(m)$ for which $x(f) \leq a$ and q is the $(X_m(a)+1)$-th node in $L(m+1)$ we have $x(e) > a$ (again, from Lemma 9.2). Node q is on the path dig(op) to the node $p = (x(e)n, n - x(e)n)$. We consider the L^∞-distance of the line segment $\overline{op}$ and q, where again $(x(e)m, m - x(e)m)$ is the L^∞-nearest point from q on $\overline{op}$. The L^∞-distance between the two points is $(x(e)m - X_m(a)) > (am - X_m(a)) > c \log n$. □

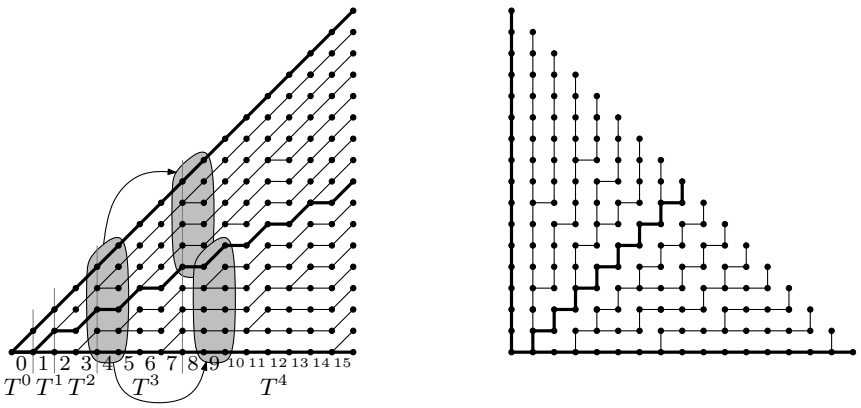
9.2.2 The Upper Bound Result

We deterministically construct a two-dimensional spanning tree DT(2) of G such that, for every $p = (i, j) \in V$, the unique path from o to p in DT(2) defines the digital ray dig(op) that represents the Euclidean line segment $\overline{op}$. By the monotonicity axiom, dig(op) is always a shortest path in the orthogonal grid.

We give the construction of DT(2) restricted to $\mathbf{G}(n)$ for $n = 2^k$. By creating rotated copies in the other quadrants and extending them to the infinite grid we get DT(2). To simplify the description (especially, when we generalize to higher dimensions later), we transform the grid by a linear map Φ that maps the lattice base $\{(1,0),(0,1)\}$ to $\{(1,0),(1,1)\}$, respectively. The linear map Φ transforms the quadrant containing $\mathbf{G}(n)$ to the the first octant and maps $\mathbf{G}(n)$ to a skew-grid with the base $\{(1,0),(1,1)\}$ in the triangular region $\{(x,y) \mid 0 \leq y \leq x \leq n\}$. The set $L(m)$ is mapped to the m-th column of the transformed grid. Figure 9.5 shows the tree T that we will construct in the skew grid as well as the corresponding tree $\Phi^{-1}(T)$ in $\mathbf{G}(n)$.

In the transformed grid $\Phi(\mathbf{G}(n))$, all edges are horizontal or diagonal with positive unit slope. An edge connecting a vertex (i, j) and a vertex $(i+1, j)$ or $(i+1, j+1)$ is called an edge in the i-th edge-column. The i-th edge-column is called an even (odd) edge-column if i is even (odd). Note that the column index starts from 0.

Since the infinite tree DT(2) cannot have leaves, the set of leaves of T restricted to $\Phi(\mathbf{G}(n))$ must be the right endpoints of the edges in the rightmost edge-column, that is, the set $\{(n, b) \mid b = 0, 1, 2, \ldots, n\}$. Any such spanning tree, and thus also the one we will construct, must satisfy the following lemma.



(a) (b)

Figure 9.5: The spanning tree $T = T^4$ (left) and the corresponding tree $\Phi^{-1}(T)$ in $\mathbf{G}(n)$ (right). The center path and the two boundary paths are highlighted in bold. The gray regions indicate how T^4 is recursively constructed by copying the columns of T^3.

Lemma 9.4 *If an edge $e \in T$ is horizontal (resp. diagonal), all the edges in T in the same edge-column below e (resp. above e) must be horizontal (resp. diagonal).*

Proof. If e is horizontal and there is a diagonal edge below e, then two edges in that column must share their right endpoint by the pigeon hole principle. This creates a cycle in T, which contradicts the fact that T is a tree. If e is diagonal a similar argument holds. □

This lemma implies that there is not much freedom for defining T, and it is also a crucial observation for generalizing the construction to higher dimensions.

We give a procedure to construct all paths from the root to the leaves of T. This suffices to define T. For sake of convenience, we denote the spanning tree restricted to the subgrid $\Phi(\mathbf{G}(2^k))$ by T^k. We have two *boundary paths*: The path towards $(2^k, 0)$ uses only horizontal edges, and the path towards $(2^k, 2^k)$ uses only diagonal edges. These are the only paths for $k = 0$ and uniquely define T^0. If $k \geq 1$, we first give the path towards $(2^k, 2^{k-1})$, which we call the *center path* (see Figure 9.5). The center path is the alternating chain of horizontal and diagonal edges, starting with the horizontal edge connecting the origin $o = (0, 0)$ and $(1, 0)$. Thus, the center path has a horizontal edge in every even column and a diagonal one in every odd column. We observe that the left endpoint of an edge of the center path in an even column is on the diagonal line $y = x/2$, while its right endpoint is below this line. The following lemma is a straightforward consequence of Lemma 9.4.

Lemma 9.5 *In the tree T^k, all the edges in an even column below the center path are horizontal and all the edges in an odd column above the center path are diagonal.*

Let's first consider the part of T^k below and including the center path. The even columns are determined by Lemma 9.5 and consist of horizontal edges only. The number of edges between the upper and lower boundary paths in the i-th column of $\Phi(\mathbf{G}(2^{k-1}))$ equals the number of edges between the center path and the lower boundary path in the $(2i + 1)$-th column of $\Phi(\mathbf{G}(2^k))$. So we can simply copy the i-th column of T^{k-1} to the lower half of the $(2i + 1)$-th column of T^k. Similarly, we know the odd columns of the part of T^k above the center path and fill the even columns by copying the i-th column

of T^{k-1} to the upper half of the $(2i)$-th column for $i = 0, 1, \ldots, 2^{k-1} - 1$. These copies do not conflict with the boundary paths of T^k. Figure 9.5a shows how column 4 in T^3 is copied to columns 8 and 9 in T^4.

This recursively constructs the tree T^k for $k \in \mathbb{N}$, and we can generate a spanning tree T of the first octant of the whole infinite grid such that T^k is the restriction of T to $\Phi(\mathbf{G}(2^k))$. Our tree in the orthogonal grid $\mathbf{G}(2^k)$ is $\Phi^{-1}(T^k)$, which we can obviously extend to DT(2), the tree on the whole orthogonal grid $\mathbf{G}$.

Theorem 9.3 *The set of digital rays defined by* DT(2) *is consistent. For any grid point* $p \in \mathbf{G}(n)$*, the* L^∞*-Hausdorff distance between* $\mathrm{dig}(op)$ *and* $\overline{op}$ *is less than* $1 + \log n$*.*

Proof. It is easy to verify that the set of digital rays defined by DT(2) is consistent, that is, it satisfies axioms (R1)–(R5). It remains to bound the distance between $\mathrm{dig}(op)$ and $\overline{op}$. Let $p = (x_p, y_p)$ be any vertex in $T = T^k$, and let $q = (x_q, y_q)$ be any vertex on $\mathrm{dig}(op)$, the path from p to o in T. We would like to claim that the vertical distance between $\overline{op}$ and q is at most k by induction on k. If $k \leq 1$, the claim is trivial. Thus, assume that the claim holds for T_{k-1}. We can further assume that $x_q \leq x_p - 2$, as we can check the claim directly otherwise.

If $\mathrm{dig}(op)$ is the center path, the claim holds by construction of the center path. Thus, we assume this is not the case. Since two paths in T cannot cross each other, both p and q must be on the same side of the center path. We distinguish the following two cases:

Case 1. If $p = (x_p, y_p)$ is below the center path (that is, $y_p < \lfloor x_p/2 \rfloor$), then $q = (x_q, y_q)$ satisfies that $y_q \leq \lfloor x_q/2 \rfloor$. From the recursive definition of T we know that the odd columns below the center path are copied from T^{k-1} and the even columns contain only horizontal edges. Thus, p is a copy of $p' = (\lfloor x_p/2 \rfloor, y_p)$, and q is a copy of $q' = (\lfloor x_q/2 \rfloor, y_q)$.

Since the claim holds for T^{k-1}, the vertical distance from q' to the line op' is at most $k-1$, that is,

$$d_y(q', op') = |y_q - y_p(\lfloor x_q/2 \rfloor)/(\lfloor x_p/2 \rfloor)| \leq k - 1.$$

Now, consider the vertical distance $d_y(q, op) = |y_q - y_p x_q / x_p|$ from q to op. We have the following inequality

$$\left| y_p \frac{\lfloor x_q/2 \rfloor}{\lfloor x_p/2 \rfloor} - y_p \frac{x_q}{x_p} \right| \leq y_p \left| \frac{x_q + 1}{x_p - 1} - \frac{x_q}{x_p} \right| = y_p \left| \frac{1}{x_p} + \frac{x_q + 1}{(x_p - 1)x_p} \right| \leq y_p \left| \frac{2}{x_p} \right| < 1 \qquad (9.1)$$

and thus

$$d_y(q, op) \leq \left| y_q - y_p \frac{\lfloor x_q/2 \rfloor}{\lfloor x_p/2 \rfloor} \right| + \left| y_p \frac{\lfloor x_q/2 \rfloor}{\lfloor x_p/2 \rfloor} - y_p \frac{x_q}{x_p} \right| \leq (k-1) + 1 = k. \qquad (9.2)$$

Case 2. If $p = (x_p, y_p)$ is above the center path (that is, $y_p > \lfloor x_p/2 \rfloor$), then $q = (x_q, y_q)$ satisfies that $y_q \geq \lfloor x_q/2 \rfloor$. The even columns above the center path are copied from T_{k-1} and the odd columns contain only diagonal edges. Thus, p is a copy of $p' = (\lfloor x_p/2 \rfloor, y_p - \lfloor x_p/2 \rfloor)$, and q is a copy of $q' = (\lfloor x_q/2 \rfloor, y_q - \lfloor x_q/2 \rfloor)$.

Since the claim holds for T^{k-1}, the vertical distance from q' to the line op' is

$$d_y(q', op') = \left| y_q - \left\lfloor \frac{x_q}{2} \right\rfloor - \left(y_p - \left\lfloor \frac{x_p}{2} \right\rfloor \right) \frac{\lfloor x_q/2 \rfloor}{\lfloor x_p/2 \rfloor} \right| = \left| y_q - y_p \frac{\lfloor x_q/2 \rfloor}{\lfloor x_p/2 \rfloor} \right| \leq k - 1, \qquad (9.3)$$

which is exactly the same expression as in Case 1. Hence, by (9.1) and the same argument as above, we get $d_y(q, op) \leq k$.

Since Φ^{-1} maps the vector $(1,0)$ to $(1,0)$ and the vector $(0,1)$ to $(-1,1)$, the L^∞-distance of q and a line op (with a positive slope) in $\mathbf{G}(n)$ is the same as the vertical distance $d_y(\Phi(q), \Phi(op))$ between the corresponding point and line in $\Phi(\mathbf{G}(n))$. Since the adjacent grid points in a digital ray have distance 1 to each other, we can analogously show that the L^∞-distance from any point on a line segment to the corresponding digital line segment is $1 + \log n$. □

Note that the tree DT(2) is related to a famous low-discrepancy sequence called the van der Corput sequence [vdC35]. Assume that n is a power of 2, and construct the sequence $X(\mathrm{DT}(2))$ using the method of Section 9.2.1 (ignoring $x_0 = 1$). Then, we obtain $X(\mathrm{DT}(2)) = (0, 1/2, 1/4, 3/4, 1/8, 5/8, 3/8, 7/8, \ldots)$, where in general for the 2-adic expansion $b_1 b_2 b_3 \ldots b_s$ of $i-1$ we have $x_i = 0.b_s b_{s-1} \ldots b_1$ for $1 \le i \le n$. This sequence is indeed the van der Corput sequence.

It is also an interesting observation that DT(2) has a quite uniform structure. Indeed, for any grid point $p = (x, y)$, the path from o to p has $\lfloor \log(|x| + |y|) \rfloor$ or $\lceil \log(|x| + |y|) \rceil$ branching vertices (excluding o) in DT(2).

9.2.3 Constant Distance Bound for Non-Monotonic Rays

Surprisingly, if we omit the monotonicity axiom (R5), the lower bound does not hold. We instead give a constant upper bound on the Hausdorff distance in Theorem 9.4. The same bound holds for the Fréchet distance if we regard a digital ray as the corresponding connected path in the graph G defining the grid topology. The digital rays that we construct are locally snake-like almost everywhere; but their bird's eye views can approximate the respective Euclidean line segments fairly well.

Theorem 9.4 *If the monotonicity axiom (R5) is not considered, there exists a system of digital rays in the plane grid such that the Hausdorff distance between each digital ray and its corresponding Euclidean line segment is $O(1)$.*

Proof. The idea is as follows: We first consider a coarser grid of width 2, and construct a spanning forest T_1 of this grid, where internal leaves are allowed. Then, we replace each node v of this forest by four nodes in the original unit-width grid such that v is located in the center of gravity of these four nodes. Then we convert the forest T_1 into a tree T_2 in the original finer grid.

Let $c > 1$ be an irrational constant. The forest T_1 is constructed as follows: We consider the *belt* $R(k) \supset \mathbf{G}(2^{k+1}) \setminus \mathbf{G}(2^k)$ defined by $2^k < x + y \le 2^{k+1}$ in the first quadrant and subdivide it into trapezoids by lines $\ell_t : y = \frac{2^k - tc}{tc} x$ passing through the non-grid points $(tc, 2^k - tc)$ on the line $x + y = 2^k$ for $t = 1, 2, \ldots, \lfloor 2^k / c \rfloor$. The widths of the two parallel edges of each trapezoid are (at most) $\sqrt{2}c$ and $2\sqrt{2}c$, respectively. Further, each trapezoid F is adjacent to one trapezoid $p(F)$ in $R(k-1)$ called the parent of F and to two trapezoids $l(F)$ and $r(F)$ in the belt $R(k+1)$ that are called the left and right child, respectively. Let q be the intersection of $x + y = 2^{k+1}$ and the dividing line of $l(F)$ and $r(F)$. The nearest grid point to q in F is called the exit node of F, and the nearest grid points to q in $l(F)$ and $r(F)$ are called their entry nodes. Each trapezoid has exactly one entry and one exit node. In Figure 9.6, the entry node and the exit node of F are marked by "E" and "X", respectively.

By gathering these trapezoids for all $k \ge \lceil \log c \rceil$, we have a decomposition of the first quadrant of the plane. Since $c > 1$, each trapezoid is wide enough so that the induced

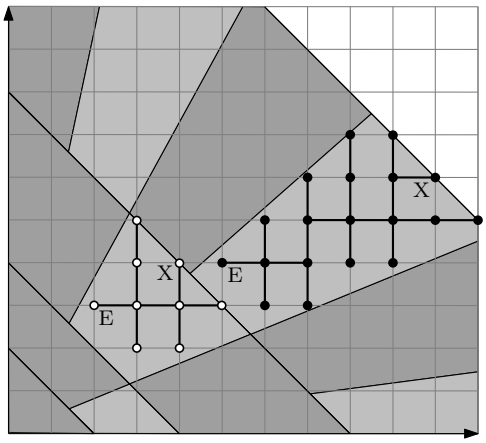


Figure 9.6: Trapezoid decomposition and two trees of the forest T_1.

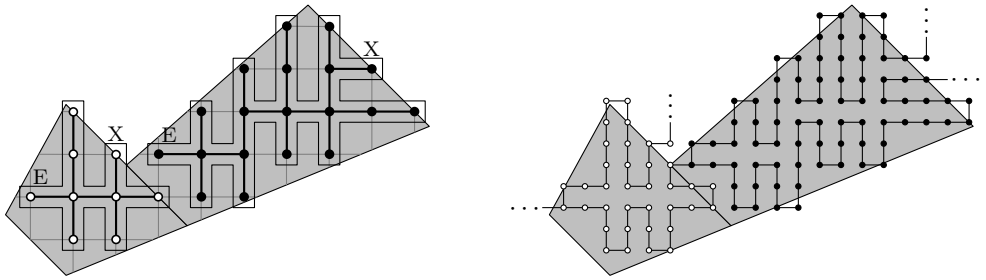


Figure 9.7: The walks around the two trees (a) and the corresponding part of the tree T_2 formed by connecting the two walks (b).

subgraph of the grid points in a trapezoid is connected. It is easy to find a spanning tree of the vertices in each trapezoid consisting of a *trunk* that is a shortest path from its entry node to its exit node, together with branches such that the length of each branch (that is, the path length from the trunk to the furthest leaf) is at most $2c$ as seen in Figure 9.6. This gives a forest T_1 consisting of small trees, one in each trapezoid. Now, let's convert T_1 into T_2 as shown in Figure 9.7. Each node of T_1 is replaced by four nodes at the corners of the surrounding unit square. Thus, we can realize the walk around the subtree of T_1 in F as a Hamiltonian cycle in the finer grid. We cut the cycle at the exit node and connect to the entry nodes of the trees in the two child trapezoids as in Figure 9.7. We obtain a tree T_2 that has no internal leaves. For any grid point $p \in F$, the line segment $\overline{op}$ is contained in the union of the ancestor trapezoids of F, and also all ancestors of p in the tree T_2 are in the same union of trapezoids. Since the width of each trapezoid is at most $2\sqrt{2}c$, the distance from any point q in the path $\mathrm{dig}(op)$ in T_2 to the line op is at most $2\sqrt{2}c$. It might happen that the nearest point from q to the line op is not in the segment $\overline{op}$ since we do not assume the monotonicity axiom. However, since the length of each branch of a subtree in T_1 is at most $2c$, the Hausdorff distance between the segment $\overline{op}$ and the path from o to p in the tree is at most $(2\sqrt{2}+2)c$.

9.3 Digital Rays in Higher-Dimensional Grids

Analogously to the two-dimensional tree DT(2), we can construct a d-dimensional tree DT(d) to define digital rays in d-dimensional space. We utilize the fact that a line in d-dimensional space is uniquely determined by its projections to all two-dimensional subspaces spanned by the first coordinate and the i-th coordinate for $i = 2, 3, \ldots, d$. We first demonstrate the construction for the case $d = 3$ and discuss the general case later.

Analogously to the two-dimensional case, we first transform the orthogonal grid by a linear map that maps the base vectors $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$ to $(1, 0, 0)$, $(1, 1, 0)$, and $(1, 1, 1)$, respectively. Thus, the first octant of the orthogonal grid is mapped to the part $\mathbf{Q}(3)$ defined by $0 \leq z \leq y \leq x$ of the skew grid spanned by three types of edges corresponding to the vectors $(1, 0, 0)$, $(1, 1, 0)$, and $(1, 1, 1)$. Next, we define a spanning tree $T(3)$ in this skew grid and transform it back to a spanning tree in the orthogonal grid.

To define $T(3)$, it suffices to define the parent of each vertex $(i, j, k) \in \mathbf{Q}(3)$. We use our previous two-dimensional tree in the skew-grid $\Phi(\mathbf{G})$, which covers the range $0 \leq y \leq x$ in the plane. We call this tree $T(2)$ implying that it is a tree in the two-dimensional skew-grid. We define two copies $T(2; x, y)$ and $T(2; x, z)$ of $T(2)$ for the dimension pairs (x, y) and (x, z) and call them the (x, y)-tree and the (x, z)-tree, respectively. The (x, y)-tree covers the range $0 \leq y \leq x$, and the (x, z)-tree covers the range $0 \leq z \leq x$.

Given a grid point $p = (i, j, k) \in \mathbf{Q}(3)$, we call p (x, y)-horizontal (resp. (x, y)-diagonal) if the edge between (i, j) and its parent in the (x, y)-tree is horizontal (resp. diagonal). Similarly, p is called (x, z)-horizontal (resp. (x, z)-diagonal) if the edge between (i, k) and its parent in the (x, z)-tree is horizontal (resp. diagonal).

The following case distinction defines the parent of p in $T(3)$:

1. if (i, j, k) is (x, y)-horizontal and (x, z)-horizontal, its parent is $(i - 1, j, k)$;
2. if (i, j, k) is (x, y)-diagonal and (x, z)-horizontal, its parent is $(i - 1, j - 1, k)$;
3. if (i, j, k) is (x, y)-diagonal and (x, z)-diagonal, its parent is $(i - 1, j - 1, k - 1)$.

There is one case missing, namely when (i, j, k) is (x, y)-horizontal and (x, z)-diagonal. The key observation is that this case cannot occur. By the definition of $\mathbf{Q}(3)$, we have $k \leq j$, and by Lemma 9.4 there is never a diagonal edge below a horizontal one in an edge column of $T(2)$. Now if (i, j, k) is (x, y)-horizontal, it must also be (x, z)-horizontal.

Therefore, we have defined a graph $T(3)$ in the grid $\mathbf{Q}(3)$, which uses only edges that are parallel to the vectors $(1, 0, 0)$, $(1, 1, 0)$, or $(1, 1, 1)$. Analogously, we can confirm that every node has at least one child. The following lemma follows from the definition of $T(3)$.

Lemma 9.6 *For every $p = (i, j, k) \in \mathbf{Q}(3)$, there is a unique path P to the origin o in $T(3)$. Thus, $T(3)$ is a tree rooted at o. The projection of P to the (x, y)-plane (resp. (x, z)-plane) coincides with the path from (i, j) (resp. (i, k)) to o in the (x, y)-tree (resp. (x, z)-tree).*

The next lemma is a consequence of Lemma 9.6 and Theorem 9.3:

Lemma 9.7 *For any plane $x = a$ where $0 \leq a \leq n$, let (a, b, c) and (a, b', c') be its intersection points with $\overline{op}$ and $\mathrm{dig}(op)$, respectively. Then, $|b - b'| < \log n$ and $|c - c'| < \log n$.*

We use the inverse map from the skew grid $\mathbf{Q}(3)$ to the three-dimensional orthogonal grid; this maps $T(3)$ to an orthogonal tree DT(3).

Proposition 9.1 *The L_1 distance from any point on the digital ray in* DT(3) *to the corresponding Euclidean line is at most $4 \log n$ if the absolute value of each coordinate value of*

the point is bounded by n. Consequently, the L_1-Hausdorff distance between a line segment and the corresponding digital ray is a most $4 \log n$.

Proof. Let's examine how the distance changes during the inverse map. The vectors $(0, 1, 0)$ and $(0, 0, 1)$ are mapped to $(-1, 1, 0)$ and $(0, -1, 1)$, respectively. Thus, a vector $(0, s, t)$ is mapped to $(-s, s - t, t)$ and $|-s| + |s - t| + |t| \leq 2|s| + 2|t|$. Thus, for $|s| \leq n$ and $|t| \leq n$, we can apply Lemma 9.7, which yields the proposition. □

For the general d-dimensional grid, we have the following theorem:

Theorem 9.5 *Given a d-dimensional grid with n^d grid points in the orthogonal topology, we can define a spanning tree $T(d)$ such that the L_1-Hausdorff distance between the line segment $\overline{op}$ and the digital ray* $\mathrm{dig}(op)$ *is less than $2(d-1)\log n$ if the absolute value of each coordinate value of p is bounded by n.*

Proof. Let $x_1, x_2, \ldots, x_d$ be the coordinates of the d-dimensional space and define $\mathbf{Q}(d)$ by $0 \leq x_d \leq x_{d-1} \leq \ldots \leq x_1$. As before, we define copies $T(2; x_1, x_i)$ of $T(2)$ for the dimension pairs (x_1, x_i), where $i = 2, 3 \ldots, d$. Now, let's consider a grid point $p = (p_1, p_2, \ldots, p_d) \in \mathbf{Q}(d)$ and define its parent in $T(d)$. By Lemma 9.4, there exists an integer $2 \leq i \leq d + 1$ such that (p_1, p_j) is diagonal in $T(2; x_1, x_j)$ for $j < i$ and horizontal for $j \geq i$. Note that all edges (p_1, p_j) are horizontal (resp. diagonal) if $i = 2$ (resp. $i = d + 1$). We connect p by an edge with the vector $(1, 1, \ldots, 1, 0, \ldots, 0, 0)$ to its parent, where the vector has $(i - 1)$ unit entries and $(d - i + 1)$ zero entries. This yields a spanning tree of the grid points of $\mathbf{Q}(d)$. The remaining analysis is analogous to the three-dimensional case. □

9.4 Mountain Approximation

In the last part of this chapter we return to an application of consistent digital rays mentioned in Section 9.1: *mountain approximation*. Recall that, geometrically, we want to approximate a noisy input terrain given as a gray-level image by a more regular terrain, in which each horizontal slice is a star-shaped region. So consider that we are given a $[0, 1]$-valued function f on the pixel grid $\mathbf{P}$. We denote f as a *pixel image function*. An important task in computer vision is to approximate f by another pixel image function ϕ from a family of functions $\mathcal{F}$ with certain desired properties. We can formulate this problem as a least-squares minimization problem, in which we aim to minimize the L_2-distance $|f - \phi|_2 = [\sum_{p \in \mathbf{P}} (f(p) - \phi(p))^2]^{1/2}$.

For any pixel image function f we define its *level set* at height t as the set $L(f, t) = \{p \in \mathbf{P} \mid f(p) \geq t\}$. A level set can be interpreted as a slice of the terrain at height t. Suppose we are given a pixel o, either by user input or automatically detected. We call f a *digital mountain function* with peak position o, if each of its level sets is a star-shaped region centered at o. In the mountain approximation problem the family $\mathcal{F}$ of pixel image functions of interest is the family of digital mountain functions with peak o. Figure 9.8 shows the approximation of an input function f by a digital mountain function ϕ.

Chen et al. [CCKT04] considered the related pyramid approximation problem, where each level set is the union of rectangles containing the peak o. Using digital star-shaped regions as level sets was left as an open problem. In fact, Chen et al. defined a more general framework of pyramid approximations for *closed region families*, that is, families of

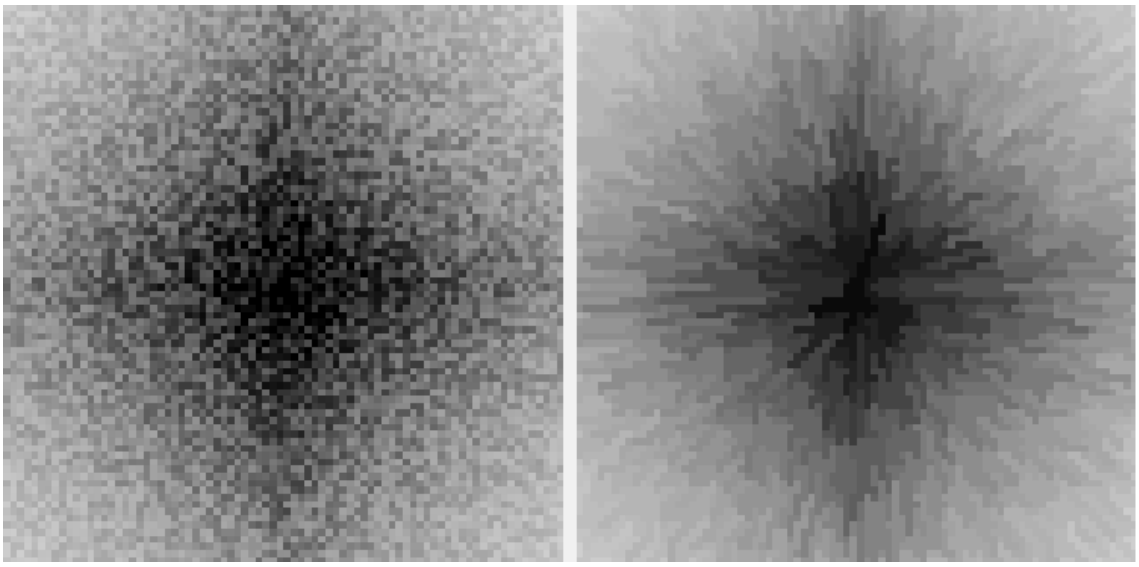


Figure 9.8: Mountain approximation: the values of the pixel image functions f (left) and ϕ (right) are represented by gray levels.

regions that are closed under intersection and union. So once we have a definition of a closed family of star-shaped regions centered at o we can closely follow their results.

A natural definition of a digital star-shaped region that comes into mind is the set of all pixels intersecting a given Euclidean star-shaped region. However, the family of star-shaped regions with a common center according to this definition is not closed under intersection and union, that is, the intersection of two digital star-shaped regions centered at o is not necessarily a digital star-shaped region again. An example is illustrated in Figure 9.9, where the intersection of two star-shaped regions is not star shaped. Instead, we use our system of consistent digital rays to define digital star-shaped regions analogously to the common definition of star-shaped regions in Euclidean space.

Definition 9.1 *Given a system of consistent digital rays from a center o, a region R is a* digital star-shaped region *centered at o if and only if* $\operatorname{dig}(op) \subseteq R$ *for any grid point $p \in R$.*

The consistency of the digital rays implies that the family of star-shaped regions with a common center according to Definition 9.1 is closed under intersection and union. Note that Definition 9.1 can naturally be extended to star shapes in higher dimensional grids. The quality of a digital star-shaped region is assured by the following theorem, which is an immediate consequence of Theorems 9.3 and 9.4.

Theorem 9.6 *There is a system of consistent digital rays with center o in $\mathbf{P}$ such that for any Euclidean star-shaped region R with center o, $R' = \bigcup_{p \in \mathbf{P} \cap R} \operatorname{dig}(op)$ is a digital star-shaped region with Hausdorff distance $H(R, R') \in O(\log n)$. Conversely, there is a system of consistent digital rays such that for any digital star-shaped region Q with center o, $Q' = \bigcup_{x \in \mathbb{R}^2 \cap Q} \overline{ox}$ is a Euclidean star-shaped region with $H(Q, Q') \in O(\log n)$. The $O(\log n)$ bound improves to $O(1)$ if we drop the monotonicity axiom.*

With the above definition of the family of digital star-shaped regions centered at o we can now apply the pyramid approximation framework of Chen et al. [CCKT04] as given in the following results.

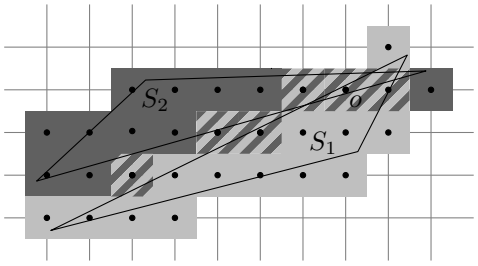


Figure 9.9: Two star-shaped regions S_1 and S_2 centered at o. The pixels of S_1 are colored in light gray, those of S_2 in dark gray. Pixels in $S_1 \cap S_2$ are bicolored. The intersection $S_1 \cap S_2$ is disconnected and thus not star shaped.

Proposition 9.2 (Chen et al. [CCKT04]) *Let $\mathcal{R}$ be a closed region family, and let $R(f,t)$ be the region in $\mathcal{R}$ maximizing $\sum_{p\in R}(f(p)-t)$ for a given pixel image function f and a real value t. If there is more than one such region, there are a maximum and a minimum (in terms of set inclusion) among them, which we denote as $R_{\max}(f,t)$ and $R_{\min}(f,t)$.*

We call t a critical height if $R_{\max}(f,t) \neq R_{\min}(f,t)$. Chen et al. [CCKT04] showed that it suffices to compute $R_{\min}(f,t)$ and $R_{\max}(f,t)$ for each critical height t in order to compute the optimal approximation ϕ.

Theorem 9.7 (Chen et al. [CCKT04]) *Let $\mathcal{F}$ be the family of pixel image functions whose level sets are regions in the closed region family $\mathcal{R}$. Then for the function $\phi \in \mathcal{F}$ that minimizes $|f-\phi|_2$, the level set $L(\phi,t) = R_{\max}(f,t)$. Moreover, $\phi(p) = t$ for a pixel $p \in \mathbf{P}$ if and only if $p \in R_{\max}(f,t) \setminus R_{\min}(f,t)$.*

Consider the family $\mathcal{S}$ of digital star-shaped regions in $\mathbf{P}$ based on the spanning tree DT(2) of G. For each vertex $v \in V$ of the tree DT(2), we define a parametric weight $w(v,t) = f(v) - t$, where $f(v)$ is the value of the input function f at the pixel corresponding to v. The pixels in the region $R(f,t)$ are spanned by a rooted subtree of DT(2) maximizing the sum of the parametric weights of the vertices. For a given t, it is easy to compute $R(f,t)$: we traverse DT(2) in a bottom-up fashion starting from the leaves and remove each vertex v and the subtree rooted at v if the sum of the parametric weights in the subtree of v (ignoring already removed vertices) is negative. The final subtree obtained by the algorithm gives $R_{\max}(f,t)$. If we replace "negative" by "non-positive" in the above procedure, we obtain $R_{\min}(f,t)$. Clearly, this can be done in linear time in terms of the tree size.

Now, we can apply a so-called *hand probing* operation: Given two heights $t_1 < t_2$, let $R_1 = R_{\max}(f,t_1)$ and $R_2 = R_{\max}(f,t_2)$, $R_1 \neq R_2$. We find the height t_3 with $t_1 < t_3 < t_2$ such that R_1 and R_2 have the same parametric weight at t_3 and compute $R_3 = R_{\max}(f,t_3)$. This operation can be done in linear time in terms of the tree size. If the height t_3 is a critical height, we define $\phi(p) = t_3$ for $p \in R_{\max}(f,t_3) \setminus R_{\min}(f,t_3)$. We recursively process the height intervals (t_1,t_3) and (t_3,t_2) and thus find all critical heights in $O(h)$ hand-probing operations, where h is the number of different heights in the input data. Note that h is bounded by $h \leq \min\{N,\Gamma\}$, where Γ denotes the number of gray levels, for example $\Gamma = 256$. In total we have a time complexity of $O(Nh)$, where N is the size of the tree DT(2), in our case $N = n^2$.

We can speed up the algorithm to run in $O(N \log h)$ time using the methods of Chen et al. [CCKT04]. Their idea is based on using a contracted tree that, for a query with heights

$t_1 < t_2$, represents only the pixels in the set $R_1 \setminus R_2$ instead of the full tree DT(2). Thus, each full level of the binary search tree can be handled in $O(N)$ time. Obviously, the number of levels of the search tree is $O(\log h)$, and thus the optimal mountain approximation can be computed in $O(N \log h)$ time. Note that if the peak position o is not specified by the user, we need to search all candidate positions for the best one.

We finally remark that the above result can be easily extended to the d-dimensional case, which is an analogue of the general pyramid construction problem considered by Chen et al. [CCKT04]. Note that the mountain approximation algorithm can also be extended to star-shaped regions based on the non-monotonic rays defined in Section 9.2.3.

9.5 Concluding Remarks

In this chapter we have investigated representations of digital line segments and, in particular, digital rays as paths in a grid topology. We have proposed a set of consistency axioms that are directly derived from the topological properties of Euclidean line segments. There is a trade-off between the consistency of a family of digital line (or ray) segments and their visual similarity to Euclidean line segments. We focused on families of digital rays from a given origin and showed a lower bound of $\Omega(\log n)$ for the worst-case Hausdorff distance between digital and Euclidean rays in an $n \times n$ grid. Subsequently we presented a simple recursive construction of a family of digital rays whose worst-case Hausdorff distance is $O(\log n)$. If the monotonicity axiom is dropped, we even achieved a constant upper bound on the Hausdorff distance. Finally, we illustrated how our consistent digital rays can be used for the mountain approximation problem in computer vision.

Open problems. Although the $O(\log n)$ bound for the Hausdorff distance is asymptotically optimal, we can improve the constant factor: The lower bound factor in discrepancy theory is merely 0.06 [Nie92]. An obviously important problem is to investigate the definition of consistent digital line segments for all pairs of grid points or, as a first step, for digital rays from multiple origins. As mentioned in Section 9.1, if the set of digital line segments satisfies the axioms, the distance bound seems to become $\Omega(n)$; it is an interesting question to prove or disprove this.

Chapter 10

Conclusion

In this thesis we have studied several network visualization problems or, more precisely, the induced graph layout problems. As Chapter 1 has indicated, there is a need for network visualizations in various disciplines. Here, we have focused on three applications: schematic public transport maps, dynamic interactive maps, and comparative drawings of binary trees, for example, in phylogenetics. The problems have been formulated as optimization problems for several aesthetic optimization criteria: mental-map preservation, bend minimization for metro lines, drawing-area minimization, crossing minimization, morphing-distance minimization, and maximization of selected labels in map labeling. For some of these problems we have designed efficient algorithms, but most of them turned out to be NP-hard. For a few of the NP-hard problems we have nonetheless been able to compute exact solutions, for example, using mixed-integer programming or branch-and bound algorithms. Otherwise, we have given approximation algorithms or heuristic methods that, at least for some of the problems, have been implemented and experimentally evaluated.

The above mentioned layout problems are not only practically relevant, but they also contain a rich structure that is equally interesting from a theoretical perspective. In addition, we have studied two graph representation problems for cover contact graphs and their complexity. The question of how to consistently represent Euclidean line segments and ray segments in a pixel grid has been studied in the last part of the thesis. There is a trade-off between visual quality and compliance with some basic topological requirements, and we obtained a tight bound for the subproblem of representing consistent rays in such a pixel grid.

Outlook

Each of the problems studied in this thesis still raises a number of interesting open problems for further research. These have been addressed in the respective chapters. Here, we want to take, to some extent, a wider perspective on the field of network visualizations. The graph drawing literature provides several multi-purpose visualization algorithms, for example, the spring embedder paradigm. Spring embedders do not make any assumptions on the properties of the graph to be drawn. This certainly makes them versatile tools for exploring unknown properties in graphs. But, on the other hand, spring embedders also cannot give any quality guarantees for the resulting layout. If available, domain knowledge of the graph at hand and its properties are a valuable piece of information that can, if incorporated into a layout algorithm, largely improve the quality of the resulting layout. Examples of such domain-specific layout algorithms have been presented in this thesis, for

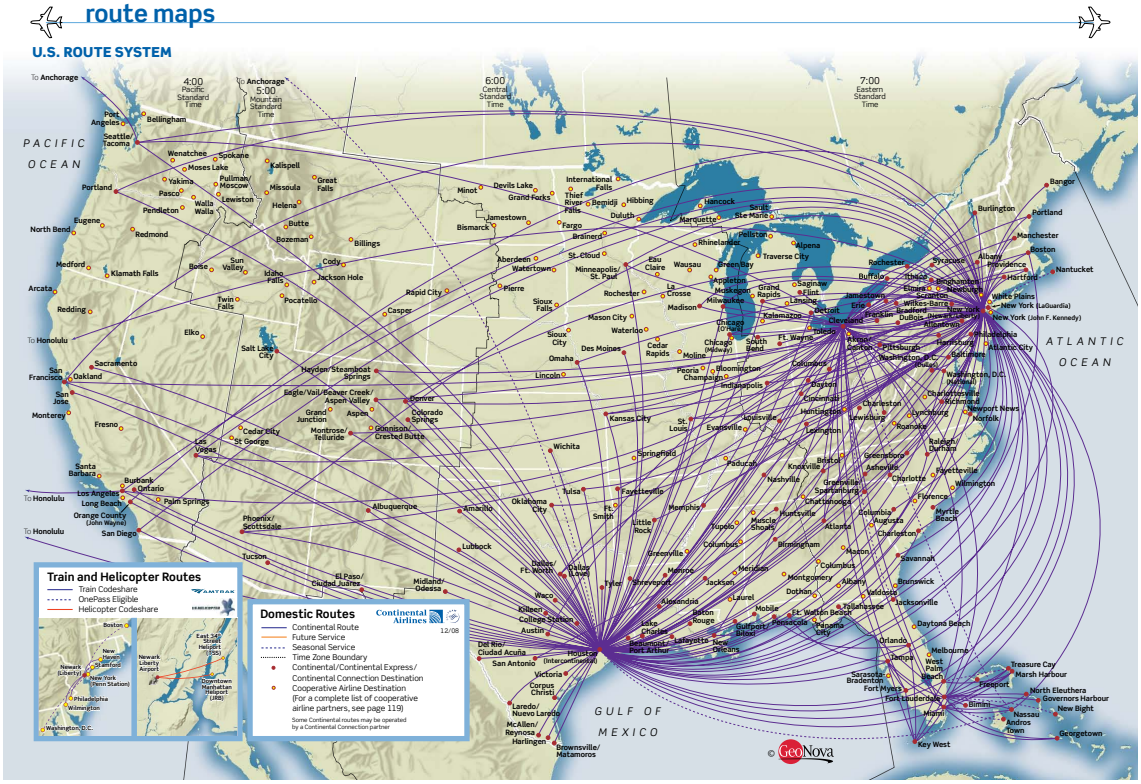


Figure 10.1: Route map of Continental Airlines.

instance, for public transport networks or for pairs of phylogenetic trees.

There is still a large need for domain-specific layout algorithms, in particular, in application areas that are just discovering that their data is in fact network data, or that layout algorithms present an interesting alternative to replace or assist the tedious manual creation of network visualizations. As an example, the domain of geographic data and cartography offers a rich pool of challenging and practically relevant graph drawing problems. Here, a distinct feature of the network data is that nodes and links are usually given with coordinates. Depending on the visualization goal, these coordinates can be modified to a certain degree in order to simplify the presentation of the network. This process is known as *generalization* in cartography.

For example, in airline route maps, direct flight connections are depicted as arcs radiating from the major hubs, see Figure 10.1. Since the actually flown routes differ from day to day depending on air traffic and weather conditions, there is no need to draw them accurately. Rather, there is a lot of flexibility, in particular, there is no embedding that needs to be preserved. What are the aesthetic constraints for such a map? The angular resolution around hub nodes is necessarily low but more important for the readability is the crossing angle of arcs. Visually following a line in the vicinity of a crossing of about 90 degrees is usually no problem, whereas one gets easily lost at crossings with very small angles. This effect has recently been observed in a user study by Huang [Hua08]. Crossing angle maximization thus seems like an interesting and important criterion for network readability—maybe more than traditional crossing minimization.

Another visualization problem arises in online route planners that generate an optimal route for each individual route request. Usually, this route is highlighted in a conventional

geographic road map. On the other hand, a schematic visualization of such a route that highlights only those route properties that are relevant for navigation such as turning directions at road intersections may be more adequate to assist a car driver. In a schematic route visualization, for example, edge directions are discretized to a small set of orientations with the main requirement that perceived turning angles at intersections are preserved. A non-uniform scale is used to depict inner-city roads and highway segments. Hand-drawn sketches of routes often follow these layout principles intuitively, and an initial study to draw route maps automatically was presented by Agrawala and Stolte [AS01]. Algorithmic approaches for schematizing individual routes on demand would complement popular route planning services on the web. Rather than creating a single elaborate map that (more or less) fits all users, the goal shifts towards creating many highly individual maps, each of which fits a single user and his specific use.

These are just two examples of applications, in which new and application-specific graph layout techniques hold great potential. They also underline the importance of the audience of network visualizations. After all, the visualizations are to be used by humans and the way we, as humans, read network visualizations is not yet systematically understood. So before designing useful layout algorithms, it is necessary to evaluate which aesthetic criteria and design rules make a visualization easy to read and understand. The positive effects on graph readability have been confirmed for classic aesthetic criteria like crossing and bend minimization [PCJ96, Pur97]. But these experiments were performed for abstract and rather small graphs, which are not necessarily representative for the networks that typically arise in an application. Once there is experimental evidence from user studies that certain criteria are more important than others for a specific use of a visualization, we can study the algorithmic implications for the corresponding graph layout and optimization problems.

There is definitely a need for more and better visualizations of networks. Graph drawing plays a key role towards that goal and it definitely remains a very attractive and interesting field for algorithmic research.

Bibliography

[ABH+06] Manuel Abellanas, Sergey Bereg, Ferran Hurtado, Alfredo García Olaverri, David Rappaport, and Javier Tejel. Moving coins. *Comput. Geom. Theory Appl.*, 34(1):35–48, 2006. [see page 177]

[ABKS09] Evmorfia Argyriou, Michael A. Bekos, Michael Kaufmann, and Antonios Symvonis. Two polynomial time algorithms for the metro-line crossing minimization problem. In I. G. Tollis and M. Patrignani, editors, *Proc. 16th Internat. Symp. Graph Drawing (GD'08)*, number 5417 in Lecture Notes Comput. Sci., pages 336–347. Springer-Verlag, 2009. [see pages 65, 66, 67, and 76]

[ACKT01] Tetsuo Asano, Danny Z. Chen, Naoki Katoh, and Takeshi Tokuyama. Efficient algorithms for optimization-based image segmentation. *Internat. J. Comput. Geom. Appl.*, 11(2):145–166, 2001. [see page 200]

[AdCC+08] Nieves Atienza, Natalia de Castro, Carmen Cortés, M. Ángeles Garrido, Clara I. Grima, Gregorio Hernández, Alberto Márquez, Auxiliadora Moreno, Martin Nöllenburg, José Ramon Portillo, Pedro Reyes, Jesús Valenzuela, Maria Trinidad Villar, and Alexander Wolff. Cover contact graphs. In S.-H. Hong, T. Nishizeki, and W. Quan, editors, *Proc. 15th Internat. Symp. Graph Drawing (GD '07)*, volume 4875 of *Lecture Notes Comput. Sci.*, pages 171–182. Springer-Verlag, 2008. [see page 173]

[AdCH+06] Manuel Abellanas, Natalia de Castro, Gregorio Hernández, Alberto Márquez, and Carlos Moreno-Jiménez. Gear system graphs. Manuscript, 2006. [see page 177]

[AGM08] Matthew Asquith, Joachim Gudmundsson, and Damian Merrick. An ILP for the metro-line crossing problem. In J. Harland and P. Manyem, editors, *Proc. 14th Computing: The Australasian Theory Symp. (CATS'08)*, volume 77 of *CRPIT*, pages 49–56. Australian Comput. Soc., 2008. [see pages 64, 65, 66, 67, 76, 77, and 81]

[AH06] Silvania Avelar and Lorenz Hurni. On the design of schematic transport maps. *Cartographica*, 41(3):217–228, 2006. [see pages 28, 30, and 31]

[AM00] Silvania Avelar and Matthias Müller. Generating topologically correct schematic maps. In *Proc. 9th Internat. Symp. Spatial Data Handling (SDH'00)*, pages 4a.28–4a.35, Beijing, 2000. [see page 29]

[Aro03] Sanjeev Arora. Approximation schemes for NP-hard geometric optimization problems: A survey. *Math. Program. Ser. B*, 97(1–2):43–69, 2003. [see page 177]

[AS01] Maneesh Agrawala and Chris Stolte. Rendering effective route maps: Improving usability through generalization. In E. Fiume, editor, *Proc. 28th Ann. Conf. Computer Graphics and Interactive Techniques (SIGGRAPH'01)*, pages 241–249. ACM, 2001. [see page 217]

[Ata99] Mikhail J. Atallah, editor. *Algorithms and Theory of Computation Handbook*. CRC Press, 1999. [see page 223]

[Ave07] Silvania Avelar. Convergence analysis and quality criteria for an iterative schematization of networks. *Geoinformatica*, 11:497–513, 2007. [see page 29]

[Ave08] Silvania Avelar. Visualizing public transport networks: An experiment in Zurich. *J. Maps*, pages 134–150, 2008. [see page 29]

[AvKS98] Pankaj K. Agarwal, Marc van Kreveld, and Subhash Suri. Label placement by maximum independent set in rectangles. *Comput. Geom. Theory Appl.*, 11(3–4):209–218, 1998. [see pages 112, 113, and 125]

[Bar80] D. J. Bartram. Comprehending spatial information: The relative efficiency of different methods of presenting information about bus routes. *J. Applied Psychology*, 65(1):103–110, 1980. [see pages 26 and 60]

[Bar03] Albert László Barabási. *Linked: How Everything Is Connected to Everything Else and What It Means for Business, Science, and Everyday Life*. Plume, 2003. [see page 1]

[BBB+09] Kevin Buchin, Maike Buchin, Jaroslaw Byrka, Martin Nöllenburg, Yoshio Okamoto, Rodrigo I. Silveira, and Alexander Wolff. Drawing (complete) binary tanglegrams: Hardness, approximation, fixed-parameter tractability. In I. G. Tollis and M. Patrignani, editors, *Proc. 16th Internat. Symp. Graph Drawing (GD'08)*, volume 5417 of *Lecture Notes Comput. Sci.*, pages 324–335. Springer-Verlag, 2009. [see page 139]

[BBdG+07] Melanie Badent, Carla Binucci, Emilio di Giacomo, Walter Didimo, Stefan Felsner, Francesco Giordano, Jan Kratochvíl, Maurizio Palladino, and Francesco Trotta. Homothetic triangle contact representations of planar graphs. In *Proc. 19th Canadian Conf. Comput. Geom. (CCCG'07)*, pages 233–236, Ottawa, Canada, 2007. [see page 176]

[BBDZ08] Joachim Böttger, Ulrik Brandes, Oliver Deussen, and Hendrik Ziezold. Map warping for the annotation of metro maps. *Computer Graphics and Applications*, 28(5):56–65, 2008. [see page 29]

[BdBMT98] Paola Bertolazzi, Giuseppe di Battista, Carlo Mannino, and Roberto Tamassia. Optimal upward planarity testing of single-source digraphs. *SIAM J. Comput.*, 27(1):132–169, 1998. [see page 143]

[BDLN05] Carla Binucci, Walter Didimo, Giuseppe Liotta, and Maddalena Nonato. Orthogonal drawings of graphs with vertex and edge labels. *Comput. Geom. Theory Appl.*, 32(2):71–114, 2005. [see pages 30, 42, and 59]

[BDY06] Ken Been, Eli Daiches, and Chee Yap. Dynamic map labeling. *IEEE Trans. Visualization and Computer Graphics*, 12(5):773–780, 2006. [see pages 109, 110, and 113]

[BE05] Ulrik Brandes and Thomas Erlebach, editors. *Network Analysis: Methodological Foundations*, volume 3418 of *Lecture Notes Comput. Sci.* Springer-Verlag, 2005. [see pages 1 and 4]

[BEKW02] Ulrik Brandes, Markus Eiglsperger, Michael Kaufmann, and Dorothea Wagner. Sketch-driven orthogonal graph drawing. In S. Kobourov and M. Goodrich, editors, *Proc. 10th Internat. Symp. Graph Drawing (GD'02)*, volume 2528 of *Lecture Notes Comput. Sci.*, pages 1–11. Springer-Verlag, 2002. [see page 28]

[Ber83] Jaques Bertin. *Semiology of Graphics: Diagrams, Networks, Maps*. Madison: University of Wisconsin Press, 1983. (French edition, 1967). [see pages 11 and 12]

[Bes02] Sergei Bespamyatnikh. An optimal morphing between polylines. *Internat. J. Comput. Geom. Appl.*, 12(3):217–228, 2002. [see page 85]

[BFN85] Carlo Batini, L. Furlani, and Enrico Nardelli. What is a good diagram? A pragmatic approach. In *Proc. 4th Internat. Conf. Entity-Relationship Approach*, pages 312–319. IEEE Computer Society, 1985. [see page 15]

[Bil05] Philip Bille. A survey on tree edit distance and related problems. *Theoret. Comput. Sci.*, 337(1–3):217–239, 2005. [see pages 140 and 141]

[BK96] Heinz Breu and David G. Kirkpatrick. On the complexity of recognizing intersection and touching graphs of disks. In F. J. Brandenburg, editor, *Proc. 3rd Internat. Symp. Graph Drawing (GD'95)*, volume 1027 of *Lecture Notes Comput. Sci.*, pages 88–98. Springer-Verlag, 1996. [see page 176]

[BK98] Heinz Breu and David G. Kirkpatrick. Unit disk graph recognition is NP-hard. *Comput. Geom. Theory Appl.*, 9(1–2):3–24, 1998. [see page 176]

[BKK+04] Adam L. Buchsbaum, Howard Karloff, Claire Kenyon, Nick Reingold, and Mikkel Thorup. OPT versus LOAD in dynamic storage allocation. *SIAM J. Comput.*, 33(3):632–646, 2004. [see page 113]

[BKPS08] Michael A. Bekos, Michael Kaufmann, Katerina Potika, and Antonios Symvonis. Line crossing minimization on metro maps. In S.-H. Hong, T. Nishizeki, and W. Quan, editors, *Proc. 15th Internat. Symp. Graph Drawing (GD'07)*, volume 4875 of *Lecture Notes Comput. Sci.*, pages 231–242. Springer-Verlag, 2008. [see pages 61, 64, 65, 66, 67, 75, and 76]

[BKR+99] Ulrik Brandes, Patrick Kenis, Jörg Raab, Volker Schneider, and Dorothea Wagner. Explorations into the visualization of policy networks. *Journal of Theoretical Politics*, 11(1):75–106, 1999. [see page 4]

[BLR00] Thomas Barkowsky, Longin Jan Latecki, and Kai-Florian Richter. Schematizing maps: Simplification of geographic shape by discrete curve evolution. In C. Freksa, W. Brauer, C. Habel, and K. F. Wender, editors, *Proc. Spatial Cognition II*, volume 1849 of *Lecture Notes Comput. Sci.*, pages 41–53. Springer-Verlag, 2000. [see page 29]

[BNPW08] Ken Been, Martin Nöllenburg, Sheung-Hung Poon, and Alexander Wolff. Optimizing active ranges for consistent dynamic map labeling. In *Proc. 24th Ann. ACM Symp. Comput. Geom. (SoCG'08)*, pages 10–19, 2008. [see page 109]

[BNPW09] Ken Been, Martin Nöllenburg, Sheung-Hung Poon, and Alexander Wolff. Optimizing active ranges for consistent dynamic map labeling. *Comput. Geom. Theory Appl.*, 2009. To appear. [see page 109]

[BNUW07] Marc Benkert, Martin Nöllenburg, Takeaki Uno, and Alexander Wolff. Minimizing intra-edge crossings in wiring diagrams and public transportation maps. In M. Kaufmann and D. Wagner, editors, *Proc. 14th Internat. Symp. Graph Drawing (GD'06)*, volume 4372 of *Lecture Notes Comput. Sci.*, pages 270–281. Springer-Verlag, 2007. [see pages 61 and 66]

[BT97] Dimitris Bertsimas and John N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997. [see page 23]

[Cc99] Bernard Chazelle and 36 co-authors. The computational geometry impact task force report. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, volume 223, pages 407–463. American Mathematical Society, 1999. [see page 113]

[CCJ90] Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete Math.*, 86(1–3):165–177, 1990. [see page 174]

[CCKT04] Danny Z. Chen, Jinhee Chun, Naoki Katoh, and Takeshi Tokuyama. Efficient algorithms for approximating a multi-dimensional voxel terrain by a unimodal terrain. In *Proc. 10th Ann. Internat. Conf. Computing and Combinatorics (COCOON'04)*, volume 3106 of *Lecture Notes Comput. Sci.*, pages 238–248. Springer-Verlag, 2004. [see pages 200, 211, 212, 213, and 214]

[CdBvD+01] Sergio Cabello, Mark de Berg, Steven van Dijk, Marc van Kreveld, and Tycho Strijk. Schematization of road networks. In *Proc. 17th Ann. ACM Symp. Comput. Geom. (SoCG'01)*, pages 33–39, 2001. [see page 29]

[CDR04] Sergio Cabello, Erik D. Demaine, and Günter Rote. Planar embeddings of graphs with specified edge lengths. In G. Liotta, editor, *Proc. 11th Internat. Symp. Graph Drawing (GD'03)*, volume 2912 of *Lecture Notes Comput. Sci.*, pages 283–294. Springer-Verlag, 2004. [see page 182]

[CEBY97] Shmuel Cohen, Gershon Elber, and Reuven Bar-Yehuda. Matching of freeform curves. *Computer-Aided Design*, 29(5):369–378, 1997. [see page 85]

[CF07] Nicholas A. Christakis and James H. Fowler. The spread of obesity in a large social network over 32 years. *New England J. Medicine*, 357(4):370–379, 2007. [see page 1]

[CG02] Alesandro Cecconi and Martin Galanda. Adaptive zooming in web cartography. *Computer Graphics Forum*, 21(4):787–799, 2002. [see page 84]

[cga] CGAL, Computational Geometry Algorithms Library. `http://www.cgal.org`. [see page 201]

[CKJ01] Jianer Chen, Iyad A. Kanj, and Weijia Jia. Vertex cover: Further observations and further improvements. *J. Algorithms*, 41:280–301, 2001. [see page 21]

[CKNT08] Jinhee Chun, Matias Korman, Martin Nöllenburg, and Takeshi Tokuyama. Consistent digital rays. In *Proc. 24th Ann. ACM Symp. Comput. Geom. (SoCG'08)*, pages 355–364, 2008. [see page 197]

[CKNT09] Jinhee Chun, Matias Korman, Martin Nöllenburg, and Takeshi Tokuyama. Consistent digital rays. *Discrete Comput. Geom.*, 2009. To appear. [see page 197]

[CLR87] Fan R. K. Chung, Frank Thomas Leighton, and Arnold L. Rosenberg. Embedding graphs in books: A layout problem with applications to VLSI design. *SIAM J. Algebraic and Discrete Methods*, 8(1):33–58, 1987. [see page 187]

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001. [see pages 11, 13, 17, 20, and 151]

[CMS95] Jon Christensen, Joe Marks, and Stuart Shieber. An empirical study of algorithms for point-feature label placement. *ACM Transactions on Graphics*, 14(3):203–232, 1995. [see page 113]

[CMS99] Stuart K. Card, Jock D. Mackinlay, and Ben Shneiderman, editors. *Readings in Information Visualization: Using Visison to Think*. Morgan Kaufmann, 1999. [see pages 3, 5, and 11]

[CR99a] Vijay Chandru and M. R. Rao. Integer programming. In Atallah [Ata99], chapter 32, pages 32/1–32/45. [see page 23]

[CR99b] Vijay Chandru and M. R. Rao. Linear programming. In Atallah [Ata99], chapter 31, pages 31/1–31/37. [see page 21]

[CS03] Charles R. Collins and Kenneth Stephenson. A circle packing algorithm. *Comput. Geom. Theory Appl.*, 25(3):233–256, 2003. [see page 174]

[CST03] Jinhee Chun, Kunihiko Sadakane, and Takeshi Tokuyama. Efficient algorithms for constructing a pyramid from a terrain. In J. Akiyama and M. Kano, editors, *Proc. Japanese Conf. on Discrete and Computational Geometry (JCDCG'02)*, volume 2866 of *Lecture Notes Comput. Sci.*, pages 108–117. Springer-Verlag, 2003. [see page 200]

[CV05] Kenneth L. Clarkson and Kasturi Varadarajan. Improved approximation algorithms for geometric set cover. In *Proc. 21st Ann. ACM Symp. Comput. Geom. (SoCG'05)*, pages 135–141, 2005. [see page 177]

[CvK03] Sergio Cabello and Marc van Kreveld. Approximation algorithms for aligning points. In *Proc. 19th Ann. ACM Symp. Comput. Geom. (SoCG'03)*, pages 20–28, 2003. [see page 29]

[dBCvKO08] Mark de Berg, Ottfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3rd edition, 2008. [see pages 129 and 135]

[dBETT99] Giuseppe di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999. [see pages 6 and 15]

[dBvKS98] Mark de Berg, Marc van Kreveld, and Stefan Schirra. Topologically correct subdivision simplification using the bandwidth criterion. *Cartography and GIS*, 25:243–257, 1998. [see page 107]

[dCCDM02] Natalia de Castro, Francisco Javier Cobos, Juan Carlos Dana, and Alberto Márquez. Triangle-free planar graphs as segment intersection graphs. *J. Graph Algorithms Appl.*, 6(1):7–26, 2002. [see page 176]

[DEG07] Michael B. Dillencourt, David Eppstein, and Michael T. Goodrich. Choosing colors for geometric graphs via color space embeddings. In M. Kaufmann and D. Wagner, editors, *Proc. 14th Internat. Symp. Graph Drawing (GD'06)*, volume 4372 of *Lecture Notes Comput. Sci.*, pages 294–305. Springer-Verlag, 2007. [see page 12]

[DEKM98] Richard Durbin, Sean Eddy, Anders Krogh, and Graeme Mitchison. *Biological Sequence Analysis*. Cambridge University Press, 1998. [see page 75]

[DF98] Rod G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1998. [see page 21]

[DFK04] Vida Dujmovič, Henning Fernau, and Michael Kaufmann. Fixed parameter algorithms for one-sided crossing minimization revisited. In G. Liotta, editor, *Proc. 11th Internat. Symp. Graph Drawing (GD'03)*, volume 2912 of *Lecture Notes Comput. Sci.*, pages 332–344. Springer-Verlag, 2004. [see page 142]

[dFOdM07] Hubert de Fraysseix and Patrice Ossona de Mendez. Representations by contact and intersection of segments. *Algorithmica*, 47(4):453–463, 2007. [see page 176]

[dFOdMP91] Hubert de Fraysseix, Patrice Ossona de Mendez, and János Pach. Representation of planar graphs by segments. *Intuitive Geometry*, 63:109–117, 1991. [see page 176]

[dFOdMR94] Hubert de Fraysseix, Patrice Ossona de Mendez, and Pierre Rosenstiehl. On triangle contact graphs. *Combin. Probab. Comput.*, 3:233–246, 1994. [see page 175]

[dFOdMR08] Hubert de Fraysseix, Patrice Ossona de Mendez, and Pierre Rosenstiehl. Representation of planar hypergraphs by contacts of triangles. In S.-H. Hong, T. Nishizeki, and W. Quan, editors, *Proc. 15th Internat. Symp. Graph Drawing (GD'07)*, volume 4875 of *Lecture Notes Comput. Sci.*, pages 125–136. Springer-Verlag, 2008. [see page 176]

[DHJ+97] Bhaskar DasGupta, Xin He, Tao Jiang, Ming Li, John Tromp, and Louxin Zhang. On distances between phylogenetic trees. In *Proc. 18th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA'97)*, pages 427–436, 1997. [see pages 140 and 141]

[dNE08] Hugo A. D. do Nascimento and Peter Eades. User hints for map labeling. *J. Visual Languages and Computing*, 19(1):39–74, 2008. [see page 30]

[DP73] David Douglas and Thomas Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10(2):112–122, 1973. [see pages 84 and 87]

[DS04] Tim Dwyer and Falk Schreiber. Optimal leaf ordering for two and a half dimensional phylogenetic tree visualization. In N. Churcher and C. Churcher, editors, *Proc. Australasian Symp. Information Visualization (invis.au'04)*, volume 35 of *CRPIT*, pages 109–115. Australian Computer Society, 2004. [see pages 143, 161, and 163]

[EET76] Gideon Ehrlich, Shimon Even, and Robert Endre Tarjan. Intersection graphs of curves in the plane. *J. Combin. Theory Ser. B*, 21(1):8–20, 1976. [see page 176]

[EHPGM01] Alon Efrat, Sariel Har-Peled, Leonidas J. Guibas, and T. M. Murali. Morphing between polylines. In *Proc. 12th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA'01)*, pages 680–689, 2001. [see pages 85 and 90]

[EJS05] Thomas Erlebach, Klaus Jansen, and Eike Seidel. Polynomial-time approximation schemes for geometric intersection graphs. *SIAM J. Comput.*, 34(6):1302–1323, 2005. [see page 112]

[EKP04] Cesim Erten, Stephen G. Kobourov, and Chandan Pitta. Intersection-free morphing of planar graphs. In *Proc. 11th Internat. Symp. Graph Drawing (GD'03)*, volume 2912 of *Lecture Notes Comput. Sci.*, pages 320–331. Springer-Verlag, 2004. [see page 85]

[Epp08] David Eppstein. Principles of graph drawing. `http://www.ics.uci.edu/~eppstein/pubs/IMBS-graph-drawing-talk.pdf`, May 2008. Talk at Institute for Mathematical Behavioral Sciences. [see page 15]

[Erd64] Paul Erdős. Problems and results on diophantine approximation. *Compos. Math.*, 16:52–65, 1964. [see page 202]

[EW94] Peter Eades and Nicholas Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11(4):379–403, 1994. [see page 142]

[FKP05] Henning Fernau, Michael Kaufmann, and Mathias Poths. Comparing trees via crossing minimization. In R. Ramanujam and S. Sen, editors, *Proc. 25th Internat. Conf. Found. Softw. Techn. Theoret. Comput. Sci. (FSTTCS'05)*, volume 3821 of *Lecture Notes Comput. Sci.*, pages 457–469. Springer-Verlag, 2005. [see pages 141, 142, 143, 144, 145, 149, 163, 170, and 171]

[For86] Steven Fortune. A sweepline algorithm for Voronoi diagrams. In *Proc. 2nd Ann. ACM Symp. Comput. Geom. (SoCG'86)*, pages 313–322, 1986. [see page 179]

[Fre00] Linton Clarke Freeman. Visualizing social networks. *Journal of Social Structure*, 1(1), 2000. [see page 4]

[FW91] Michael Formann and Frank Wagner. A packing problem with applications to lettering of maps. In *Proc. 7th Ann. ACM Symp. Comput. Geom. (SoCG'91)*, pages 281–288, 1991. [see page 113]

[Gar94] Ken Garland. *Mr Beck's Underground Map*. Capital Transport Publishing, 1994. [see pages 26, 30, 55, 58, and 62]

[GCV+07] Kwang-Il Goh, Michael E. Cusick, David Valle, Barton Childs, Marc Vidal, and Albert-László Barabási. The human disease network. *Proc. Natl. Acad. Sci. USA*, 104(21):8685–8690, 2007. [see pages 1, 2, and 3]

[GDCV99] Jonas Gomes, Lucia Darsa, Bruno Costa, and Luiz Velho. *Warping and Morphing of Graphical Objects*. Morgan Kaufmann, 1999. [see page 84]

[GG03] S. Guindon and O. Gascuel. A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Systematic Biology*, 52(5):696–704, 2003. [see page 164]

[GGHT97] Michael T. Goodrich, Leonidas J. Guibas, John Hershberger, and Paul J. Tanenbaum. Snap rounding line segments efficiently in two and three dimensions. In *Proc. 13th Ann. ACM Symp. Comput. Geom. (SoCG'97)*, pages 284–293, 1997. [see page 200]

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979. [see pages 11, 17, 18, and 22]

[GJ83] Michael R. Garey and David S. Johnson. Crossing number is NP-complete. *SIAM J. Algebraic and Discrete Methods*, 4(3):312–316, 1983. [see page 67]

[GPQX07] Carsten Görg, Mathias Pohl, Ermir Qeli, and Kai Xu. Visual representations. In A. Kerren, A. Ebert, and J. Meyer, editors, *Human-Centered Visualization Environments*, volume 4417 of *Lecture Notes Comput. Sci.*, chapter 4, pages 163–230. Springer-Verlag, 2007. [see page 11]

[GPS89] Jacob E. Goodman, Richard Pollack, and Bernd Sturmfels. Coordinate representation of order types requires exponential storage. In *Proc. 21st Ann. ACM Symp. Theory Comput. (STOC'89)*, pages 405–410, 1989. [see page 200]

[GT83] Harold N. Gabow and Robert Endre Tarjan. A linear-time algorithm for a special case of disjoint set union. In *Proc. 15th Ann. ACM Symp. Theory Comput. (STOC'83)*, pages 246–251, 1983. [see page 151]

[GT96] Ashim Garg and Roberto Tamassia. A new minimum cost flow algorithm with applications to graph drawing. In S. North, editor, *Proc. 4th Internat. Symp. Graph Drawing (GD'96)*, volume 1190 of *Lecture Notes Comput. Sci.*, pages 201–216. Springer-Verlag, 1996. [see pages 16 and 33]

[GT01] Ashim Garg and Roberto Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM J. Comput.*, 31(2):601–625, 2001. [see page 16]

[GW95] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, 1995. [see pages 142, 156, and 157]

[GY86] Daniel H. Greene and F. Frances Yao. Finite-resolution computational geometry. In *Proc. 27th Ann. IEEE Symp. Foundations Comput. Sci. (FOCS'86)*, pages 143–152, 1986. [see page 200]

[Hal80] William K. Hale. Frequency assignment: Theory and applications. *Proc. IEEE*, 68(12):1497–1514, 1980. [see page 174]

[HJLM93] Frank Harary, Michael S. Jacobson, Marc J. Lipman, and Fred R. McMorris. Abstract sphere-of-influence graphs. *Math. Comput. Modelling*, 17(11):77–83, 1993. [see page 180]

[HK01] Petr Hliněný and Jan Kratochvíl. Representing graphs by disks and balls (a survey of recognition-complexity results). *Discrete Math.*, 229(1–3):101–124, 2001. [see page 176]

[Hli00] Petr Hliněný. *Contact Representations of Graphs*. PhD thesis, Charles University Prague, Faculty of Mathematics and Physics, 2000. [see page 176]

[Hli01] Petr Hliněný. Contact graphs of line segments are NP-complete. *Discrete Math.*, 235(1–3):95–106, 2001. [see page 176]

[HM85] Dorit S. Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *J. ACM*, 32:130–136, 1985. [see pages 112, 123, 124, and 176]

[HMdN06] Seok-Hee Hong, Damian Merrick, and Hugo A. D. do Nascimento. Automatic visualization of metro maps. *J. Visual Languages and Computing*, 17(3):203–224, 2006. [see pages 29, 30, 31, 34, 40, 43, 49, 50, 52, 53, and 55]

[HMM00] Ivan Herman, Guy Melan con, and M. Scott Marshall. Graph visualization and navigation in information visualization: a survey. *IEEE Trans. Visualization and Computer Graphics*, 6(1):24–43, 2000. [see page 6]

[Hro03] Juraj Hromkovic. *Algorithmics for Hard Problems*. Springer-Verlag, 2nd edition, 2003. [see page 19]

[HSV+94] Mark S. Hafner, Philip D. Sudman, Francis X. Villablanca, Theresa A. Spradling, James W. Demastes, and Steven A. Nadler. Disparate rates of molecular evolution in cospeciating hosts and parasites. *Science*, 265(5175):1087–1090, 1994. [see page 140]

[HT74] John Hopcroft and Robert Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, 1974. [see page 176]

[Hua08] Weidong Huang. An eye tracking study into the effects of graph layout. Arxiv report, October 2008. Available at `http://arxiv.org/abs/0810.4431`. [see pages 171 and 216]

[HvW08] Danny Holten and Jarke J. van Wijk. Visual comparison of hierarchically organized data. In *Proc. 10th Eurographics/IEEE-VGTC Symp. Visualization (EuroVis'08)*, pages 759–766, 2008. [see pages 143, 161, and 162]

[HW02] William C. Hahn and Robert A. Weinberg. A subway map of cancer pathways, 2002. Poster in Nature Reviews Cancer. [see pages 2, 4, and 27]

[Inf07] Information Architects. Web trend map 2007 version 2.0. `http://informationarchitects.jp/ia-trendmap-2007v2`, 2007. [see page 27]

[Jen06] Bernhard Jenny. Geometric distortion of schematic network maps. *SoC Bulletin*, 40:15–18, 2006. [see page 29]

[JLM95] Michael S. Jacobson, Marc J. Lipman, and Fred R. McMorris. Trees that are sphere-of-influence graphs. *Appl. Math. Lett.*, 8:89–93, 1995. [see page 180]

[JM97] Michael Jünger and Petra Mutzel. 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. *J. Graph Algorithms Appl.*, 1(1):1–25, 1997. [see pages 30 and 142]

[JW05] Christopher B. Jones and J. Mark Ware. Map generalization in the web age. *Internat. J. Geographical Information Sci.*, 19(8–9):859–870, 2005. [see page 84]

[Kar84] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984. [see page 21]

[Kho02] Subhash Khot. On the power of unique 2-prover 1-round games. In *Proc. 34th Ann. ACM Symp. Theory Comput. (STOC'02)*, pages 767–775, 2002. [see page 144]

[Kic08] Kick Design. The kick map. `http://www.kickmap.com`, November 2008. [see page 5]

[KM94] Jan Kratochvíl and Jiří Matoušek. Intersection graphs of segments. *J. Combin. Theory Ser. B*, 62(2):289–315, 1994. [see page 176]

[KM99a] Gunnar W. Klau and Petra Mutzel. Combining graph labeling and compaction. In J. Kratochvíl, editor, *Proc. 8th Internat. Symp. Graph Drawing (GD'99)*, volume 1731 of *Lecture Notes Comput. Sci.*, pages 27–37. Springer-Verlag, 1999. [see pages 17 and 30]

[KM99b] Gunnar W. Klau and Petra Mutzel. Optimal compaction of orthogonal grid drawings. In G. P. Cornuéjols, R. E. Burkard, and G. J. Woeginger, editors, *Proc. 7th Conf. on Integer Programming and Combinat. Optimization (IPCO'99)*, volume 1610 of *Lecture Notes Comput. Sci.*, pages 304–319. Springer-Verlag, 1999. [see page 30]

[Koe36] Paul Koebe. Kontaktprobleme der konformen Abbildung. *Ber. Sächs. Akad. Wiss. Leipzig, Math.-Phys. Klasse*, 88:141–164, 1936. [see pages 173, 175, 176, and 182]

[KR92] Donald E. Knuth and Arvind Raghunathan. The problem of compatible representatives. *SIAM J. Discr. Math.*, 5(3):422–427, 1992. [see pages 18, 113, 117, and 121]

[KR04] R. Klette and A. Rosenfeld. Digital straightness—a review. *Discrete Appl. Math.*, 139(1–3):197–230, 2004. [see pages 197 and 198]

[Kre02] Valdis Krebs. Uncloaking terrorist networks. *First Monday*, 7(4), 2002. [see page 1]

[KV05] Subhash Khot and Nisheeth K. Vishnoi. The unique games conjecture, integrality gap for cut problems and embeddability of negative type metrics into l_1. In *Proc. 46th Ann. IEEE Symp. Foundations Comput. Sci. (FOCS'05)*, pages 53–62, 2005. [see pages 141 and 144]

[KW01] Michael Kaufmann and Dorothea Wagner, editors. *Drawing Graphs: Methods and Models*, volume 2025 of *Lecture Notes Comput. Sci.* Springer-Verlag, 2001. [see pages 6 and 15]

[KW02] Michael Kaufmann and Roland Wiese. Embedding vertices at points: Few bends suffice for planar graphs. *J. Graph Algorithms Appl.*, 6(1):115–129, 2002. [see pages 177, 181, 186, and 187]

[LCR+06] Heng Li, Avril Coghlan, Jue Ruan, Lachlan James Coin, Jean-Karim Hériché, Lara Osmotherly, Ruiqiang Li, Tao Liu, Zhang Zhang, Lars Bolund, Gane Ka-Shu Wong, Weimou Zheng, Paramvir Dehal, Jun Wang, and Richard Durbin. TreeFam: a curated database of phylogenetic trees of animal gene families. *Nucleic Acids Research*, 34:D572–D580, 2006. [see page 164]

[Len90] Thomas Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* John Wiley & Sons, 1990. [see page 63]

[Li06] Heng Li. *Constructing the TreeFam Database.* PhD thesis, Institute of Theoretical Physics, Chinese Academy of Science, 2006. [see page 164]

[Lic82] David Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11(2):329–343, 1982. [see pages 18, 113, 182, and 193]

[LJLH05] Fran cois Lecordix, Yolène Jahard, Cécile Lemarié, and Etienne Hauboin. The end of carto 2001 project: Top100 based on bdcarto database. In *Proc. 8th ICA Workshop on Generalisation and Multiple Representation*, A Coruña, Spain, July 2005. [see page 92]

[Low89] David Lowe. Organization of smooth image curves at multiple scales. *Internat. J. Computer Vision*, 3(2):119–130, 1989. [see page 88]

[LPR+08] Antoni Lozano, Ron Y. Pinter, Oleg Rokhlenko, Gabriel Valiente, and Michal Ziv-Ukelson. Seeded tree alignment. *IEEE/ACM Trans. Comput. Biology and Bioinformatics*, 5(4):503–513, 2008. [see pages 141 and 143]

[LS02] Ulrich Lauther and Andreas Stübinger. Generating schematic cable plans using springembedder methods. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Proc. 10th Internat. Symp. Graph Drawing (GD'01)*, volume 2265

of *Lecture Notes Comput. Sci.*, pages 465–466. Springer-Verlag, 2002. [see page 29]

[Mac95] Alan M. MacEachren. *How Maps Work: Representation, Visualization, and Design*. The Guilford Press, 1995. [see page 11]

[Mat99] Jiří Matoušek. *Geometric Discrepancy: An Illustrated Guide*. Springer-Verlag, 1999. [see page 202]

[MELS95] Kazuo Misue, Peter Eades, Wei Lai, and Kozo Sugiyama. Layout adjustment and the mental map. *J. Visual Languages and Computing*, 6(2):183–210, 1995. [see page 85]

[MG07] Damian Merrick and Joachim Gudmundsson. Path simplification for metro map layout. In M. Kaufmann and D. Wagner, editors, *Proc. 14th Internat. Symp. Graph Drawing (GD'06)*, volume 4372 of *Lecture Notes Comput. Sci.*, pages 258–269. Springer-Verlag, 2007. [see page 29]

[Mit92] Joseph S. B. Mitchell. L_1 shortest paths among polygonal obstacles in the plane. *Algorithmica*, 8(1–6):55–88, 1992. [see page 178]

[MM99] Terry A. McKee and F. R. McMorris. *Topics in Intersection Graph Theory*. SIAM Monographs on Discrete Mathematics. SIAM, 1999. [see page 175]

[MN90] Kurt Mehlhorn and Stefan Näher. Dynamic fractional cascading. *Algorithmica*, 5(1–4):215–241, 1990. [see page 136]

[MNWB07] Damian Merrick, Martin Nöllenburg, Alexander Wolff, and Marc Benkert. Morphing polygonal lines: A step towards continuous generalization. In A. Winstanley, editor, *Proc. 15th Ann. Geograph. Inform. Sci. Research Conf. UK (GISRUK'07)*, pages 390–399, Maynooth, Ireland, 2007. [see page 86]

[Mor53] J. L. Moreno. *Who Shall Survive?* Beacon House, 1953. [see page 4]

[Mor96] Alastair Morrison. Public transport maps in western European cities. *Cartographic J.*, 33(2):93–110, 1996. [see page 26]

[MPN+05] Lauren Ancel Meyers, Babak Pourbohloul, M. E. J. Newman, Danuta M. Skowronski, and Robert C. Brunham. Network theory and SARS: predicting outbreak diversity. *J. Theoretical Biology*, 232(1):71–81, 2005. [see page 1]

[MRS07] William A. Mackaness, Anne Ruas, and L. Tiina Sarjakoski, editors. *Generalisation of Geographic Information: Cartographic Modelling and Applications*. Elsevier, 2007. [see page 84]

[MSS95] M. Marek-Sadowska and M. Sarrafzadeh. The crossing distribution problem. *IEEE Trans. Computer-Aided Design*, 14(4):423–433, 1995. [see page 67]

[MUV02] Xavier Muñoz, Walter Unger, and Imrich Vrt'o. One sided crossing minimization is NP-hard for sparse graphs. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Proc. 9th Internat. Symp. Graph Drawing (GD'01)*, volume 2265 of *Lecture Notes Comput. Sci.*, pages 115–123. Springer-Verlag, 2002. [see page 142]

[Nag05] Hiroshi Nagamochi. An improved bound on the one-sided minimum crossing number in two-layered drawings. *Discrete Comput. Geom.*, 33(4):565–591, 2005. [see page 142]

[NBW06] Mark Newman, Albert-László Barabási, and Duncan J. Watts. *The Structure and Dynamics of Networks*. Princeton University Press, 2006. [see page 1]

[Nes04] Keith V. Nesbitt. Getting to more abstract places using the metro map metaphor. In *Proc. 8th Internat. Conf. Information Visualisation (IV'04)*, pages 488–493. IEEE, 2004. [see page 27]

[Ney99] Gabriele Neyer. Line simplification with restricted orientations. In F. K. Dehne, A. Gupta, J.-R. Sack, and R. Tamassia, editors, *Proc. 6th Internat. Workshop Algorithms and Data Structures (WADS'99)*, volume 1663 of *Lecture Notes Comput. Sci.*, pages 13–24. Springer-Verlag, 1999. [see page 29]

[Nie92] Harald Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*, volume 63 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM, 1992. [see pages 202 and 214]

[Nie06] Rolf Niedermeier. *Invitation to Fixed Parameter Algorithms*. Oxford University Press, 2006. [see page 21]

[NK00] Masatoshi Nei and Sudhir Kumar. *Molecular Evolution and Phylogenetics*. Oxford University Press, 2000. [see page 140]

[NMWB08] Martin Nöllenburg, Damian Merrick, Alexander Wolff, and Marc Benkert. Morphing polylines: A step towards continuous generalization. *Computers, Environment and Urban Systems*, 32(4):248–260, 2008. [see page 83]

[Nöl05] Martin Nöllenburg. Automated drawing of metro maps. Technical Report 2005-25, Fakultät für Informatik, Universität Karlsruhe, 2005. Available at `http://digbib.ubka.uni-karlsruhe.de/volltexte/1000004123`. [see pages 28 and 33]

[Nöl07] Martin Nöllenburg. Geographic visualization. In A. Kerren, A. Ebert, and J. Meyer, editors, *Human-Centered Visualization Environments*, volume 4417 of *Lecture Notes Comput. Sci.*, chapter 6, pages 257–294. Springer-Verlag, 2007. [see pages 83 and 109]

[NR04] Takao Nishizeki and Md. Saidur Rahman. *Planar Graph Drawing*, volume 12 of *Lecture Notes Series on Computing*. World Scientific, 2004. [see pages 6 and 15]

[NVWH09] Martin Nöllenburg, Markus Völker, Alexander Wolff, and Danny Holten. Drawing binary tanglegrams: An experimental evaluation. In *Proc. 11th Workshop Algorithm Engineering and Experiments (ALENEX'09)*, New York, 2009. SIAM. To appear. [see page 139]

[NW] Martin Nöllenburg and Alexander Wolff. Drawing and labeling high-quality metro maps by mixed-integer programming. *IEEE Trans. Visualization and Computer Graphics*. Submitted and under revision. [see page 25]

[NW06] Martin Nöllenburg and Alexander Wolff. A mixed-integer program for drawing high-quality metro maps. In P. Healy and N. S. Nikolov, editors, *Proc. 13th Internat. Symp. Graph Drawing (GD'05)*, volume 3843 of *Lecture Notes Comput. Sci.*, pages 321–333. Springer-Verlag, 2006. [see page 25]

[O'R03] O'Reilly. Open source route map. `http://www.oreilly.de/artikel/routemap.pdf`, 2003. [see page 27]

[Ove03] Mark Ovenden. *Metro Maps of the World*. Capital Transport Publishing, 2003. [see pages 26, 28, 31, 44, 62, 64, and 81]

[PA95] János Pach and Pankaj K. Agarwal. *Combinatorial Geometry*. John Wiley and Sons, New York, 1995. (Contains a proof of Koebe's theorem.). [see page 173]

[PAF95] Corinne Plazanet, Jean-Georges Affholder, and Emmanuel Fritsch. The importance of geometric modeling in linear feature generalization. *Cartography and Geographic Information Systems*, 22(4):291–305, 1995. [see page 86]

[Pag02] Roderic D. M. Page, editor. *Tangled Trees: Phylogeny, Cospeciation, and Coevolution*. University of Chicago Press, 2002. [see pages 139, 140, and 141]

[PCJ96] Helen C. Purchase, Robert F. Cohen, and Murray James. Validating graph drawing aesthetics. In F. J. Brandenburg, editor, *Proc. 3rd Internat. Symp. Graph Drawing (GD'95)*, volume 1027 of *Lecture Notes Comput. Sci.*, pages 435–446. Springer-Verlag, 1996. [see pages 15 and 217]

[PCJ97] Helen C. Purchase, Robert F. Cohen, and Murray I. James. An experimental study of the basis for graph drawing algorithms. *J. Experimental Algorithms*, 2:Article 4, 1997. [see pages 62 and 141]

[PGP03] Ingo Petzold, Gerhard Gröger, and Lutz Plümer. Fast screen map labeling—data-structures and algorithms. In *Proc. 23rd Internat. Cartographic Conf. (ICC'03)*, pages 288–298, Durban, South Africa, 2003. ICA. [see page 113]

[PPH99] Ingo Petzold, Lutz Plümer, and Markus Heber. Label placement for dynamically generated screen maps. In *Proc. 19th Internat. Cartographic Conf. (ICC'99)*, pages 893–903, Ottawa, Canada, 1999. ICA. [see page 113]

[PPS90] János Pach, Richard Pollack, and Joel Spencer. Graph distance and Euclidean distance on the grid. In R. Bodendiek and R. Henn, editors, *Topics in Graph Theory and Combinatorics*, pages 555–559. Physica-Verlag, 1990. [see page 199]

[PS05] Sheung-Hung Poon and Chan-Su Shin. Adaptive zooming in point set labeling. In M. Liśkiewicz and R. Reischuk, editors, *Proc. 15th Internat. Symp. Fundam. Comput. Theory (FCT'05)*, volume 3623 of *Lecture Notes Comput. Sci.*, pages 233–244. Springer-Verlag, 2005. [see page 113]

[Pur97] Helen Purchase. Which aesthetic has the greatest effect on human understanding? In G. di Battista, editor, *Proc. 5th Internat. Symp. Graph Drawing (GD'97)*, volume 1353 of *Lecture Notes Comput. Sci.*, pages 248–261. Springer-Verlag, 1997. [see pages 62, 139, 141, and 217]

[PW01] János Pach and Rephael Wenger. Embedding planar graphs at fixed vertex locations. *Graphs Combin.*, 17(4):717–728, 2001. [see page 177]

[Rob05] Maxwell J. Roberts. *Underground Maps After Beck*. Capital Transport, 2005. [see pages 28, 31, and 55]

[RRR98] Venkatesh Raman, B. Ravikumar, and S. Srinivasa Rao. A simplified NP-complete MAXSAT problem. *Inform. Process. Lett.*, 65(1):1–6, 1998. [see page 145]

[Sac94] Horst Sachs. Coin graphs, polyhedra, and conformal mapping. *Discrete Math.*, 134(1-3):133–138, 1994. [see page 174]

[SB04] Monika Sester and Claus Brenner. Continuous generalization for visualization on small mobile devices. In P. F. Fisher, editor, *Proc. 11th Internat. Symp. Spatial Data Handling (SDH'04)*, pages 355–368. Springer-Verlag, 2004. [see page 84]

[Sch72] Wolfgang M. Schmidt. Irregularities of distribution, VII. *Acta Arithmetica*, 21:45–50, 1972. [see page 202]

[Sch77] Wolfgang M. Schmidt. *Lectures on Irregularities of Distribution*. Tata Inst. Fund. Res. Bombay, 1977. [see page 202]

[Sch84] Edward R. Scheinerman. *Intersection classes and multiple intersection parameters of graphs*. PhD thesis, Princeton University, 1984. [see page 176]

[Sch86] Alexander Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1986. [see page 23]

[Sch88] Philip J. Schneider. Phoenix: An interactive curve design system based on the automatic fitting of hand-sketched curves. Master's thesis, Department of Computer Science, University of Washington, 1988. [see page 87]

[Sch95] Otfried Schwarzkopf. The extensible drawing editor Ipe. In *Proc. 11th Ann. ACM Symp. Comput. Geom. (SoCG'95)*, pages C10–C11, 1995. [see page 201]

[SE98] Tatiana Samoilov and Gershon Elber. Self-intersection elimination in metamorphosis of two-dimensional curves. *The Visual Computer*, 14(8–9):415–428, 1998. [see page 85]

[Sez01] Tevfik Metin Sezgin. Feature point detection and curve approximation for early processing of free-hand sketches. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 2001. [see pages 86 and 87]

[SG01a] Vitaly Surazhsky and Craig Gotsman. Controllable morphing of compatible planar triangulations. *ACM Trans. Graphics*, 20(4):1–21, 2001. [see page 85]

[SG01b] Vitaly Surazhsky and Craig Gotsman. Morphing stick figures using optimized compatible triangulations. In *Proc. 9th Pacific Conf. Computer Graphics and Applications (PG'01)*, pages 40–49. IEEE, 2001. [see pages 85, 86, and 107]

[SGSK01] Elmer S. Sandvad, Kaj Grønbæk, Lennert Sloth, and Jørgen Lindskov Knudsen. A metro map metaphor for guided tours on the Web: the Webvise Guided Tour System. In *Proc. 10th Internat. World Wide Web Conf. (WWW'01)*, pages 326–333. ACM, 2001. [see page 27]

[Shn92] Ben Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Trans. Graphics*, 11(1):92–99, 1992. [see page 17]

[Shn96] Ben Shneiderman. The eyes have it: A task by data taxonomy for information visualizations. In *Proc. IEEE Symp. Visual Languages*, pages 336–343, 1996. [see page 5]

[SN87] Naruya Saitou and Masatoshi Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4):406–425, 1987. [see page 164]

[SR04] Jonathan M. Stott and Peter Rodgers. Metro map layout using multicriteria optimization. In *Proc. 8th Internat. Conf. Information Visualisation (IV'04)*, pages 355–362. IEEE, 2004. [see pages 30, 31, 34, 40, 43, 49, 50, 52, and 55]

[SR05] Jonathan M. Stott and Peter Rodgers. Automatic metro map design techniques. In *Proc. 22nd Internat. Cartographic Conf. (ICC'05)*, A Coruña, Spain, 2005. ICA. [see pages 30, 49, 50, 52, and 55]

[SRB+05] Jonathan M. Stott, Peter Rodgers, Remo Aslak Burkhard, Michael Meier, and Matthias Thomas Jelle Smis. Automatic layout of project plans using a metro map metaphor. In *Proc. 9th Internat. Conf. Information Visualisation (IV'05)*, pages 203–206. IEEE, 2005. [see page 27]

[Sto08] Jonathan Stott. *Automatic Layout of Metro Maps Using Multicriteria Optimisation.* PhD thesis, University of Kent, Computing Laboratory, 2008. [see pages 30, 49, 50, 55, and 60]

[STT81] Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Systems, Man, and Cybernetics*, 11(2):109–125, 1981. [see pages 15, 142, 143, and 162]

[Sug01] Kokichi Sugihara. Robust geometric computation based on topological consistency. In V. N. Alexandrov, J. Dongarra, B. A. Juliano, R. S. Renner, and C. J. K. Tan, editors, *Proc. Internat. Conf. Computational Science, Part 1 (ICCS'01)*, volume 2073 of *Lecture Notes Comput. Sci.*, pages 12–26. Springer-Verlag, 2001. [see page 200]

[Syd08] Sydney CityRail. `http://www.cityrail.nsw.gov.au/networkmaps/network_map.pdf`, 2008. [see pages 31 and 32]

[Tam87] Roberto Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987. [see pages 16 and 33]

[Tei02] Steven L. Teig. The X Architecture: not your father's diagonal wiring. In *Proc. Internat. Workshop on System-Level Interconnect Prediction (SLIP'02)*, pages 33–37. ACM, 2002. [see page 28]

[Thu80] William P. Thurston. *The Geometry and Topology of 3-Manifolds*. Princeton University Notes, 1980. [see page 174]

[Tou88] Godfried T. Toussaint. A graph-theoretical primal sketch. In Godfried T. Toussaint, editor, *Computational Morphology: A Computational Geometric Approach to the Analysis of Form*, pages 229–260. North-Holland, 1988. [see page 180]

[Tra08] Transport for London. `http://www.tfl.gov.uk/gettingaround/1106.aspx`, 2008. [see page 56]

[Tuf90] Edward R. Tufte. *Envisioning Information*. Graphics Press, 1990. [see pages 3, 11, and 12]

[Tuf97] Edward R. Tufte. *Visual Explanations: Images and Quantities, Evidence and Narrative*. Graphics Press, 1997. [see pages 3 and 11]

[Tuf01] Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, 2nd edition, 2001. [see pages 2, 3, 11, and 60]

[Vaz01] Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001. [see pages 20 and 177]

[vdC35] Johannes van der Corput. Verteilungsfunktionen I & II. *Nederl. Akad. Wetensch. Proc.*, 38:813–820, 1058–1066, 1935. [see page 208]

[vK01] Marc van Kreveld. Smooth generalization for continuous zooming. In *Proc. 20th Internat. Cartographic Conf. (ICC'01)*, pages 2180–2185. ICA, 2001. [see page 84]

[vKSW99] Marc van Kreveld, Tycho Strijk, and Alexander Wolff. Point labeling with sliding labels. *Comput. Geom. Theory Appl.*, 13(1):21–47, 1999. [see pages 113 and 122]

[vOV04] René van Oostrum and Remco C. Veltkamp. Parametric search made practical. *Comput. Geom. Theory Appl.*, 28(2–3):75–88, 2004. [see page 90]

[War04] Colin Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann, 2004. [see page 11]

[Wat03] Duncan J. Watts. *Six Degrees: The Science of a Connected Age*. W W Norton & Co., 2003. [see page 1]

[WC02] Xiaodong Wu and Danny Z. Chen. Optimal net surface problems with applications. In *Proc. 29th Internat. Colloquium Automata, Languages and Programming (ICALP'02)*, volume 2380 of *Lecture Notes Comput. Sci.*, pages 1029–1042. Springer-Verlag, 2002. [see page 200]

[WD99] Robert Weibel and Geoffrey Dutton. Generalising spatial data and dealing with multiple representations. In Paul A. Longley, Michael F. Goodchild, David J. Maguire, and David W. Rhind, editors, *Geographical Information Systems – Principles and Technical Issues*, volume 1, chapter 10, pages 125–155. John Wiley & Sons, 1999. [see page 84]

[Wel91] Emo Welzl. Smallest enclosing disks (balls and ellipsoids). In H. Maurer, editor, *New Results and New Trends in Computer Science*, volume 555 of *Lecture Notes Comput. Sci.*, pages 359–370. Springer-Verlag, 1991. [see page 176]

[Wie08] Wiener Linien. `http://www.wienerlinien.at/media/files/2008/SVP_Deutsch_3288.pdf`, 2008. [see page 46]

[Wil01] Thomas Willhalm. Software packages. In M. Kaufmann and D. Wagner, editors, *Drawing Graphs: Methods and Models*, volume 2025 of *Lecture Notes Comput. Sci.*, pages 274–281. Springer-Verlag, 2001. [see page 16]

[Wol07] Alexander Wolff. Drawing subway maps: A survey. *Informatik – Forschung und Entwicklung*, 22(1):23–44, 2007. [see page 29]

[WS96] Alexander Wolff and Tycho Strijk. The Map-Labeling Bibliography. `http://i11www.ira.uka.de/map-labeling/bibliography`, 1996. [see page 113]

[Wu06] Xiaodong Wu. Efficient algorithms for the optimal-ratio region detection problems in discrete geometry with applications. In T. Asano, editor, *Proc. 17th Internat. Symp. Algorithms and Computation (ISAAC'06)*, volume 4288 of *Lecture Notes Comput. Sci.*, pages 289–299. Springer-Verlag, 2006. [see page 200]

[ZC06] Wan Nazmee Wan Zainon and Paul Calder. Visualising phylogenetic trees. In W. Piekarski, editor, *Proc. 7th Austalasian User Interface Conf. (AUIC'06)*, volume 50 of *CRPIT*, pages 145–152. Australian Comput. Soc., 2006. [see page 143]

Index

List of Publications

Book Chapter

[1] **Geographic visualization**. In: Andreas Kerren, Achim Ebert, and Joerg Meyer, editors, *Human-Centered Visualization Environments*, volume 4417 of *Lecture Notes Comput. Sci.*, chapter 6, pages 257–294. Springer-Verlag, 2007. [see pages 83 and 109]

Journal Articles

[2] **Optimizing active ranges for consistent dynamic map labeling**. *Comput. Geom. Theory Appl.*, 2009. Special issue of SoCG'08, to appear. Joint work with Ken Been, Sheung-Hung Poon, and Alexander Wolff. [see page 109]

[3] **Consistent digital rays**. *Discrete Comput. Geom.*, 2009. Special issue of SoCG'08, to appear. Joint work with Jinhee Chun, Matias Korman, and Takeshi Tokuyama. [see page 197]

[4] **Algorithms for multi-criteria boundary labeling**. *J. Graph Algorithms Appl.*, 2009. Special issue of GD'07, to appear. Joint work with Marc Benkert, Herman Haverkort, and Moritz Kroll.

[5] **Morphing polylines: A step towards continuous generalization**. *Computers, Environment and Urban Systems*, 32(4):248–260, 2008. Special issue of GISRUK'07. Joint work with Damian Merrick, Alexander Wolff, and Marc Benkert. [see page 83]

[6] **Drawing and labeling high-quality metro maps by mixed-integer programming**. *IEEE Trans. Visualization and Computer Graphics*. Submitted and under revision. Joint work with Alexander Wolff. [see page 25]

Articles in Refereed Conference Proceedings

[7] **Drawing binary tanglegrams: An experimental evaluation**. In: *Proc. 11th Workshop Algorithm Engineering and Experiments (ALENEX'09)*, pages 106–119, New York, 2009. SIAM. Joint work with Markus Völker, Alexander Wolff, and Danny Holten. [see page 139]

[8] **Drawing (complete) binary tanglegrams: Hardness, approximation, fixed-parameter tractability**. In: I. G. Tollis and M. Patrignani, editors, *Proc. 16th Internat. Symp. Graph Drawing (GD'08)*, volume 5417 of *Lecture Notes Comput. Sci.*, pages 324–335. Springer-Verlag, 2009. Joint work with Kevin Buchin, Maike Buchin, Jaroslaw Byrka, Yoshio Okamoto, Rodrigo I. Silveira, and Alexander Wolff. [see page 139]

[9] **Optimizing active ranges for consistent dynamic map labeling**. In: *Proc. 24th Ann. ACM Symp. Comput. Geom. (SoCG'08)*, pages 10–19, 2008. Joint work with Ken Been, Sheung-Hung Poon, and Alexander Wolff. [see page 109]

[10] **Consistent digital rays**. In: *Proc. 24th Ann. ACM Symp. Comput. Geom. (SoCG'08)*, pages 355–364, 2008. Joint work with Jinhee Chun, Matias Korman, and Takeshi Tokuyama. [see page 197]

[11] **Boundary labeling with octilinear leaders**. In: J. Gudmundsson, editor, *Proc. 11th Scandinavian Workshop on Algorithm Theory (SWAT'08)*, volume 5124 of *Lecture Notes Comput. Sci.*, pages 234–245. Springer-Verlag, 2008. Joint work with Michael A. Bekos, Michael Kaufmann, and Antonios Symvonis.

[12] **Algorithms for multi-criteria one-sided boundary labeling**. In: S.-H. Hong and T. Nishizeki, editors, *Proc. 15th Internat. Symp. Graph Drawing (GD'07)*, volume 4875 of *Lecture Notes Comput. Sci.*, pages 243–254. Springer-Verlag, 2008. Joint work with Marc Benkert, Herman Haverkort, and Moritz Kroll.

[13] **Cover contact graphs**. In: S.-H. Hong and T. Nishizeki, editors, *Proc. 15th Internat. Symp. Graph Drawing (GD'07)*, volume 4875 of *Lecture Notes Comput. Sci.*, pages 171–182. Springer-Verlag, 2008. Joint work with Nieves Atienza, Natalia de Castro, Carmen Cortés, M. Ángeles Garrido, Clara I. Grima, Gregorio Hernández, Alberto Márquez, Auxiliadora Moreno, José Ramon Portillo, Pedro Reyes, Jesús Valenzuela, Maria Trinidad Villar, and Alexander Wolff. [see page 173]

[14] **Morphing polygonal lines: A step towards continuous generalization**. In: A. Winstanley, editor, *Proc. 15th Ann. Geographic Information Sci. Research Conf. UK (GISRUK'07)*, pages 390–399, Maynooth, Ireland, 11–13 April 2007. Joint work with Damian Merrick, Alexander Wolff, and Marc Benkert. [see page 86]

[15] **Minimizing intra-edge crossings in wiring diagrams and public transportation maps**. In: M. Kaufmann and D. Wagner, editors, *Proc. 14th Internat. Symp. Graph Drawing (GD'06)*, number 4372 in Lecture Notes Comput. Sci., pages 270–281. Springer-Verlag, 2007. Joint work with Marc Benkert, Takeaki Uno, and Alexander Wolff. [see pages 61 and 66]

[16] **A mixed-integer program for drawing high-quality metro maps**. In: P. Healy and N. S. Nikolov, editors, *Proc. 13th Internat. Symp. Graph Drawing (GD'05)*, volume 3843 of *Lecture Notes Comput. Sci.*, pages 321–333. Springer-Verlag, 2006. Joint work with Alexander Wolff. [see page 25]

Articles in Non-Refereed Workshop Proceedings

[17] **Improved algorithms for length-minimal one-sided boundary labeling**. In: *Proc. 23rd European Workshop Comput. Geom. (EuroCG'07)*, pages 190–193, Graz, Austria, 19–21 March 2007. Joint work with Marc Benkert.

[18] **Validation in the cluster analysis of gene expression data**. In: R. Mikut and M. Reischl, editors, *Proc. 14th Workshop Fuzzy-Systeme und Computational Intelligence*, pages 13–32, Dortmund, Germany, November 2004. GI Fachgruppe Fuzzy-Systeme und Soft-Computing, Universitätsverlag Karlsruhe. Joint work with Jens Jäkel.

Thesis

[19] **Automated drawing of metro maps**. Master's thesis, Fakultät für Informatik, Universität Karlsruhe (TH), August 2005.

Curriculum Vitæ

Name	Martin Nöllenburg
Date of Birth	28 June 1979
Place of Birth	Heilbronn
Nationality	German

06/1998	Abitur (university entrance qualification), Gymnasium Neckargemünd
08/1998–08/1999	Alternative civilian service as emergency medical technician, Malteser Hilfsdienst Wiesloch
10/1999	Enrollment as a student in Informatics at Universität Karlsruhe (TH)
09/2002–05/2003	Visiting student at McGill University, Montréal, Canada supported by a scholarship of the German Academic Exchange Service (DAAD)
08/2005	Diploma in Informatics, Universität Karlsruhe (TH)
10/2005–02/2009	Ph.D. student and research assistant in the project "Geometric Networks and their Visualization" funded by the German Research Foundation (DFG), Fakultät für Informatik, Universität Karlsruhe (TH). Advisor: PD Dr. Alexander Wolff
29.01. – 23.02.2007	Research guest of Prof. Takeshi Tokuyama at Tohoku University, Sendai, Japan and Dr. Takeaki Uno, National Institute of Informatics, Tokyo, Japan
10.09. – 21.09.2007	Research guest of Dr. Joachim Gudmundsson, NICTA, Sydney, Australia