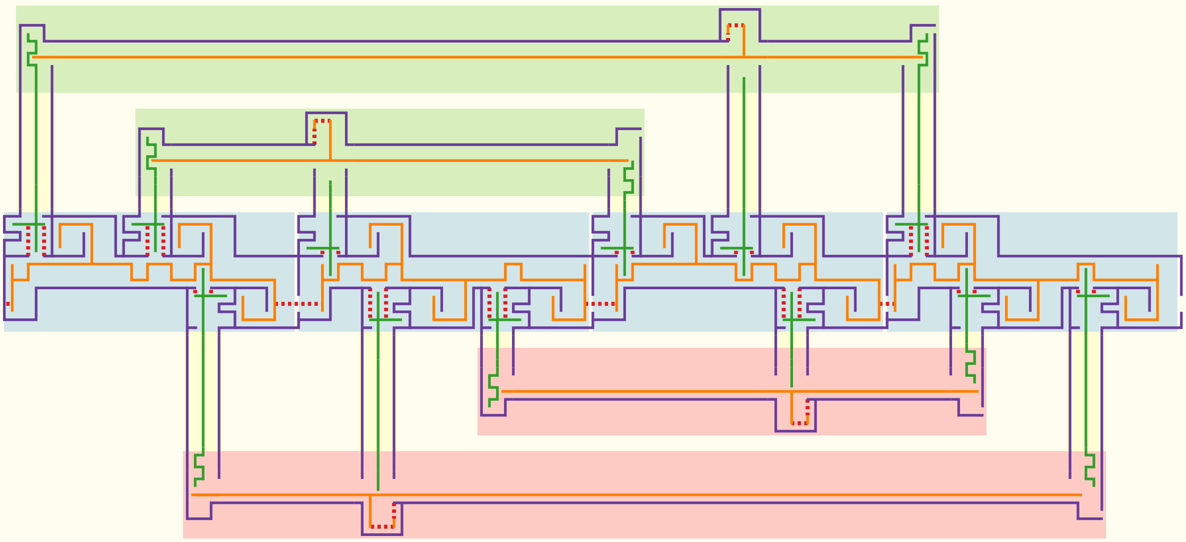
Johannes Zink

Algorithms for Drawing Graphs and Polylines with Straight-Line Segments



Johannes Zink

Algorithms for Drawing Graphs and Polylines with Straight-Line Segments

Johannes Zink

Algorithms for Drawing Graphs and Polylines with Straight-Line Segments

Dissertation
Julius-Maximilians-Universität Würzburg

Dissertation, Julius-Maximilians-Universität Würzburg
Fakultät für Mathematik und Informatik, 2023
Gutachter: Prof. Dr. Alexander Wolff, Prof. Dr. Martin Nöllenburg, Prof. Dr. Michael Kaufmann

Impressum:

Johannes Zink
Am Heerweg 14
D-97725 Langendorf

Abgabe von Exemplaren an:

Universitätsbibliothek Würzburg
Julius-Maximilians-Universität Würzburg
Am Hubland
D-97074 Würzburg

Vorwort und Danksagung

Zuerst und besonders möchte ich meinem Doktorvater, Alexander Wolff – oder, wenn man ihn etwas kennt, kurz Sascha – danken. Tatsächlich hätte ich mich ohne ihn vermutlich nicht für ein Promotionsstudium entschieden und dementsprechend niemals eine Doktorarbeit geschrieben. Als ich gerade dabei war, meine Masterarbeit unter der Betreuung von ihm, Steven Chaplick und Fabian Lipp zu schreiben, kam Sascha auf mich zu und fragte mich, ob ich nicht promovieren wolle. Bis dahin war das nicht mein Plan, aber die Idee reizte mich – während meiner Masterarbeit hatte ich schon einen kleinen, aber positiven Eindruck von sowohl der Forschung im Bereich des Graphzeichnens als auch dem Klima in der Algorithmik-Arbeitsgruppe bekommen – und so sagte ich Sascha in meinem jugendlichen Leichtsinn zu.

Heute würde ich rückblickend sagen, dass es eine sehr gute Entscheidung war. Ich habe interessante Einblicke in die Forschung und in die Lehre bekommen, die mir als Informatikstudent noch weitestgehend verschlossen geblieben sind. Das gilt natürlich mehr für die Forschung als für die Lehre – so konnte ich ein wenig erleben wie der wissenschaftliche Betrieb vonstattengeht, wie Peer-Reviewing abläuft und wie man auf Konferenzen und Workshops die Menschen hinter den kleinen und großen Namen auf den Papern tatsächlich kennenlernt. Hier möchte ich es Sascha hoch anrechnen, dass er mich beim Forschen sehr früh an die Hand genommen hat, man sich praktisch jederzeit an ihn wenden konnte, um mit ihm Rücksprache zu halten, und er sich auch dafür eingesetzt hat, dass ich zu Konferenzen fahren und an wissenschaftlichen Workshops teilnehmen konnte. Zudem geht mein Dank an Sabine Storandt, mit der ich zu Beginn meines Promotionsstudiums während Saschas Forschungsfreisemesters gemeinsam forschen durfte. Basierend darauf haben wir noch Jahre später an gemeinsamen Forschungsprojekten gearbeitet, selbst als sie Würzburg schon verlassen und eine Professur in Konstanz angenommen hatte.

Viele meiner Forschungsergebnisse konnte ich in diese Doktorarbeit einbringen. Das wäre in dieser Form nicht ohne meine Koautoren der zugehörigen Journal- und Konferenzartikel möglich gewesen, denen ich hiermit allen danken möchte. Ich danke – neben meinem Doktorvater – auch Martin Nöllenburg und Michael Kaufmann für das Begutachten dieser Doktorarbeit und Marie Schmidt und Martin Nöllenburg für das Prüfen bei meiner Disputation am 8.5.2023.

Aber auch in der Lehre habe ich positive Erfahrungen gemacht. Man hört manchmal eine Variation des Satzes *man hat ein Thema erst wirklich verstanden, wenn man es anderen beigebracht hat.* Ich würde sagen, dass da auch etwas dran ist. Übungen zu halten, Übungsblätter zu korrigieren, sich Aufgaben ausdenken und zuletzt auch Vorlesungen zu halten bringt einen dazu, sich auch auf die Details eines Themas einzulassen und die Materie wirklich zu verinnerlichen. Grundsätzlich haben mir die Themen aus der Informatik in Lehre und Forschung und das mathematische Denken fürs Leben tiefere Einblicke und Einsichten gebracht, um die ich sehr dankbar bin. Auch hierfür hat es sich für mich gelohnt, sich nach dem Master noch weit-

erzubilden und zu promovieren. Davon abgesehen konnte ich auch einige praktische Fähigkeiten ausbauen – sei es im Bereich des wissenschaftlichen Schreibens mit LaTeX, des Präsentierens (ja, ipe ist in vielen Punkten das deutlich bessere Programm als PowerPoint) und ein Stück weit auch des Programmierens. Besonders wichtig für funktionierende Forschung und Lehre ist meines Erachtens eine gute Zusammenarbeit mit den Kollegen. Bei allen, die hier über die Jahre gekommen und gegangen sind, möchte ich mich an dieser Stelle ausdrücklich bedanken. Ich habe beim Weg in die Mensa, bei der Arbeit und auch darüber hinaus die immer sehr positive Atmosphäre und auch manch gutes Gespräch genossen.

Doch es braucht nicht nur ein gutes Arbeitsumfeld, sondern auch im Privaten hilft es Menschen zu haben, auf die man sich verlassen kann und die einen unterstützen. Hier möchte ich mich bei meinen Freunden, Bekannten und Verwandten bedanken, die immer zu mir standen und mit denen ich manche schöne Zeit verbringen konnte, um einen angenehmen Ausgleich aus Arbeit, Promotion, Hobby und Freizeit zu haben. Ein besonderer Dank geht hier an meine Freundin Lara, die mich in den wichtigen Phasen der Doktorarbeit immer unterstützt hat und auch ein bisschen Probe gelesen hat. Zuletzt möchte ich mich bei meiner Familie, meinen drei Geschwistern, Eva-Maria, Christina und Leonhard, und ganz besonders meinen Eltern, Klaus und Elisabeth, bedanken. Ihr habt mir von klein auf ein schönes und behütetes Umfeld geboten, in dem ich gut aufwachsen konnte und in das ich auch immer wieder gerne zurückkomme. Dies war die Basis für alles Weitere.

Johannes Zink im Frühjahr 2024

Zusammenfassung

Graphen stellen ein wichtiges Mittel dar, um Beziehungen zwischen Objekten zu modellieren. Sie bestehen aus *Knoten*, die die Objekte repräsentieren, und *Kanten*, die Beziehungen zwischen Paaren von Objekten abbilden. Um Menschen die Struktur eines Graphen zu vermitteln, ist es nahezu unumgänglich den Graphen zu visualisieren. Eine solche Visualisierung nennen wir *Graphzeichnung.* Eine Graphzeichnung ist *geradlinig*, wenn jeder Knoten als ein Punkt (oder ein kleines geometrisches Objekt, z. B. ein Rechteck) und jede Kante als eine Strecke zwischen ihren beiden Knoten dargestellt ist. Eine sehr einfache geradlinige Graphzeichnung, bei der alle Knoten eine Folge bilden, entlang der die Knoten durch Kanten verbunden sind, nennen wir *Polylinie.* Ein Beispiel für eine Polylinie in der Praxis ist eine GPS-Trajektorie. Das zugrundeliegende Straßennetzwerk wiederum kann als Graph repräsentiert werden.

In diesem Buch befassen wir uns mit Fragen, die sich bei der Arbeit mit geradlinigen Graphzeichnungen und Polylinien stellen. Insbesondere untersuchen wir Algorithmen zum Erkennen von bestimmten mit Strecken darstellbaren Graphen, zum Generieren von geradlinigen Graphzeichnungen und zum Abstrahieren von Polylinien.

Im ersten Teil schauen wir uns zunächst an, wie und in welcher Zeit wir entscheiden können, ob ein gegebener Graph ein *Stickgraph* ist, das heißt, ob sich seine Knoten als vertikale und horizontale Strecken auf einer diagonalen Geraden darstellen lassen, die sich genau dann schneiden, wenn zwischen ihnen eine Kante liegt. Anschließend betrachten wir die *visuelle Komplexität* von Graphen. Konkret untersuchen wir für bestimmte Graphklassen, wie viele Strecken für jede geradlinige Graphzeichnung notwendig sind, und, ob drei (oder mehr) verschiedene Streckensteigungen ausreichend sind, um alle Kanten zu zeichnen. Zuletzt beschäftigen wir uns mit der Frage, wie wir den Knoten eines Graphen mit gerichteten und ungerichteten Kanten (geordnete) Farben zuweisen können, sodass keine benachbarten Knoten dieselbe Farbe haben und Farben entlang gerichteter Kanten aufsteigend sind. Hierbei ist die spezielle Eigenschaft der betrachteten Graphen, dass sich die Knoten als Intervalle darstellen lassen, die sich genau dann überschneiden, wenn eine Kanten zwischen ihnen verläuft.

Das letztgenannte Problem ist motiviert von einer Anwendung beim automatisierten Zeichnen von Kabelplänen mit vertikalen und horizontalen Streckenverläufen, womit wir uns im zweiten Teil befassen. Wir beschreiben einen Algorithmus, welcher die abstrakte Beschreibung eines Kabelplans entgegennimmt und daraus eine Zeichnung generiert, welche die speziellen Eigenschaften dieser Kabelpläne, wie Stecker und Gruppen von zusammengehörigen Drähten, berücksichtigt. Anschließend evaluieren wir die Qualität der so erzeugten Zeichnungen experimentell.

Im dritten Teil befassen wir uns mit dem Abstrahieren bzw. Vereinfachen einer einzelnen Polylinie und eines Bündels von Polylinien. Bei diesem Problem sollen aus einer oder mehreren gegebenen Polylinie(n) so viele Knoten wie möglich entfernt werden, wobei jede resultierende Polylinie ihrem ursprünglichen Verlauf (nach einem gegeben Maß) hinreichend ähnlich bleiben muss.

Abstract

Graphs provide a key means to model relationships between entities. They consist of *vertices* representing the entities, and *edges* representing relationships between pairs of entities. To make people conceive the structure of a graph, it is almost inevitable to visualize the graph. We call such a visualization a *graph drawing*. Moreover, we have a *straight-line* graph drawing if each vertex is represented as a point (or a small geometric object, e.g., a rectangle) and each edge is represented as a line segment between its two vertices. A *polyline* is a very simple straight-line graph drawing, where the vertices form a sequence according to which the vertices are connected by edges. An example of a polyline in practice is a GPS trajectory. The underlying road network, in turn, can be modeled as a graph.

This book addresses problems that arise when working with straight-line graph drawings and polylines. In particular, we study algorithms for recognizing certain graphs representable with line segments, for generating straight-line graph drawings, and for abstracting polylines.

In the first part, we first examine, how and in which time we can decide whether a given graph is a *stick graph*, that is, whether its vertices can be represented as vertical and horizontal line segments on a diagonal line, which intersect if and only if there is an edge between them. We then consider the *visual complexity* of graphs. Specifically, we investigate, for certain classes of graphs, how many line segments are necessary for any straight-line graph drawing, and whether three (or more) different slopes of the line segments are sufficient to draw all edges. Last, we study the question, how to assign (ordered) colors to the vertices of a graph with both directed and undirected edges such that no neighboring vertices get the same color and colors are ascending along directed edges. Here, the special property of the considered graph is that the vertices can be represented as intervals that overlap if and only if there is an edge between them.

The latter problem is motivated by an application in automated drawing of cable plans with vertical and horizontal line segments, which we cover in the second part. We describe an algorithm that gets the abstract description of a cable plan as input, and generates a drawing that takes into account the special properties of these cable plans, like plugs and groups of wires. We then experimentally evaluate the quality of the resulting drawings.

In the third part, we study the problem of abstracting (or *simplifying*) a single polyline and a bundle of polylines. In this problem, the objective is to remove as many vertices as possible from the given polyline(s) while keeping each resulting polyline sufficiently similar to its original course (according to a given similarity measure).

Contents

Chapter 1
Introduction

Algorithms are ubiquitous in our world. We often associate them with the rise of computers and the Internet in the last decades, which is still in progress. And to some degree this is true: computers actually run many different algorithms, computers have made us aware of algorithms, computers have improved our understanding of algorithms, and computers have made us intensify our research on algorithms – computer science as an own discipline is comparably young. However, algorithms are more fundamental than computers. We can presume them to be part of mathematics, providing a tool to describe instructions and procedures.

For instance, we have investigated sorting algorithms because sorting is a basic problem that often needs to be performed in computer programs. However, ever since people group in large settlements and form complex societies, they have had things to sort. Maybe without being conscious of it, they have applied some sorting algorithm. This could have been a version of bucket sort. You first group, say, sheets with names by their initial letter onto piles and then you order each pile independently using the insertion sort algorithm. There, you iteratively move names to their correct positions within the pile. The way librarians sort books has even motivated the design of a new sorting algorithm called *library sort* [BFM06, wik22c], which is a version of insertion sort with gaps. (As it is beneficial to leave gaps in a shelf for books encountered later on to avoid moving too many books.) There are many other examples of algorithms being established for centuries. A recipe to bake a cake and the rules for manual division in school are essentially algorithms.

Also, nature and evolution are full of algorithms. If some behavior (which may be seen as an algorithm) is more energy efficient than another behavior with the same outcome, then evolution selects for the former and the more efficient algorithm spreads. A famous example of evolutionary evolved algorithmic behavior in nature are ant colonies applying algorithms for finding paths [JSKC11], building bridges [RLP+15], or fighting over territory [HL80]. Their behavior has parallels in systems like the Internet [SN22] and, in computer science, it has motivated the concept of *ant colony optimization algorithms* [wik22a]. In some sense, these algorithms are hard-coded in the genes of the ants. Of course, not every algorithm is discovered by random mutations and natural selection, but simple ones are favored, which is why so many symmetric structures occur in living things [JDG+22]. This is because evolution itself can be understood as an algorithmic process. Besides evolution, also our universe follows specific rules and hence allows – at least to some degree – being described and simulated mathematically and algorithmically [MHS+22].

A *problem* is a concept closely tied to algorithms. Algorithms usually do not exist for an end in themselves, but to solve a problem. A central aspect of computer science is to study problems from an algorithmic and computational perspective. This book is actually more about the analysis and the algorithmic complexity of some problems in the domain of graph and polyline drawing than it is purely about algorithms therein.

A fundamental insight that computer science has told us, is that we can classify problems by how efficiently they can be solved. We remark that this classification usually refers to the growth of the running time required to solve a problem when we increase the input size. However, scaling problems and their instances is a natural thing that we encounter almost everywhere in our world. Our world is a place of multiple levels of encapsulation: quarks form hadrons that form atoms that form molecules that form organelles that form cells that form tissues that form organs that form human beings that form families that form communities that form cities that form states that form a globalized network. Somewhat similarly, a key property of computer programs is to allow encapsulation. Hence, knowing the behavior of an algorithm used as a subroutine when scaling the input makes sense.

To an outside person, it may sometimes sound impressive that we can prove that – under common assumptions – many practical problems have the property that they cannot always be solved efficiently, but if we have a solution, we can always verify efficiently that it is correct. Such problems can be routing vehicles along shortest round tours in a network of cities, scheduling tasks to people and machines, playing tetris [BDH^{+}04], or packing your knapsack [wik22d]. Such asymmetries, which are sometimes hard to understand, also occur in cryptographic systems, where we can prove, based on somewhat older number-theoretic knowledge, some degree of safety – again under some common assumptions. Under the hood of our computers and smartphones, we use cryptographic systems every day, making them one among many great examples why it is worth doing theory and mathematics – beyond the gain of insights about the world and the joy of seeing elegant proofs, artful constructions, and fascinating connections.

Something else our modern world has made us more aware of is that there are many networks all around us: computer networks, company supply networks, social networks, highway networks, criminal networks, public transport networks, metabolic networks, and many more. The mathematical term for network is *graph*. A graph consists of a set of objects, which we call *vertices* or sometimes *nodes*, and a set of relations between pairs of vertices, which we call *edges*. In discrete mathematics and computer science, a graph is a vital structure that is well-studied in many different aspects. Still many problems in graph theory remain open.

To come back to real-world networks, not only can they be modeled as graphs, but also many questions and problems concerning such networks can be formulated as graph problems. For instance, finding the most influential people in a social or criminal network corresponds to finding the central vertices in a graph. Setting up optimal bus tours in a city network corresponds to finding shortest paths and round tours in a graph. A problem from graph theory that we consider in this book is *graph coloring* (Chapter 6). There, one is given a graph, and the task is to assign colors to the vertices such that no two neighboring vertices share the same color. The objective is to minimize the number of different colors. This abstract problem can often be used to model real-world problems such as assigning frequency bands to radio stations while avoiding interferences between stations with overlapping transmission ranges, or scheduling school classes to time slots such that each teacher holds no more than one lesson at a time, or – even more directly – assigning distinct colors to neighboring countries on a map.

When humans operate on networks, like computer network managers analyzing link traffic or railroad administrators monitoring trains or police officers detecting criminal activities in a bank transaction network, it is often beneficial to have a good visualization of the network at hand. Several studies [PMCC01, PCA02, WPCM02, HHE08, PSD09, PHNK12, KMLM16] (see also a recent survey by Burch et al. [BHW$^+$21]) have shown that graph layout aesthetics, quality metrics, and the drawing style have significant influence on the readability of a graph and how well people perform when they are asked to solve specific tasks on the drawn graphs.

Problems and studies like these have motivated the field of *graph drawing*. In computer science, we understand this term primarily as studying the algorithmic aspects of generating graph drawings and the mathematical properties of graph drawings and their existence. Graph drawing research covers the complete range from purely theoretical to highly applied questions and results. Also in this book, we present a small selection of topics from the realm of graph drawing. Most of it is of theoretical nature, but there is also a chapter that describes our findings in computing and evaluating cable plan drawings that are to be used in industry. However, even there we analyze the theoretical background and we notice that, as a subproblem, there occurs a graph coloring problem, which we solve with an algorithm from one of the theory chapters of this book. In Parts I and II, we consider graph drawing problems.

An area related to graph drawing is *computational geometry*. It is about algorithms and algorithmic complexity in the context of geometric structures. Graphs can be used as a tool, and drawn graphs can also be the geometric objects being processed. Hence, it overlaps with graph drawing. However, besides graphs, many geometric structures like point sets, line segments, planes, polygons, polyhedra, and spheres are processed; see the *Handbook of Discrete and Computational Geometry* [GOT18] for more information. Note that computational geometry has many applications in computer graphics, robotics, geographic information systems, and many more. In Part III, we consider a problem from computational geometry.

When leaving an integer grid and working with real-numbered coordinates – maybe because we use circles, or line segments with specific slopes – this raises numerical issues of how precise our computations need to be in practical applications, which makes implementations more complicated. Also in mathematics, specifically in geometry, there are many generalizations of the classical two- or three-dimensional Euclidean geometry. This may include high-dimensional spaces (which are hard to imagine for a human being), curved spaces or geometry on surfaces of other geometric objects. Doubtlessly these generalizations have their right to exist and dozens of times they have proven to have real-world applications, however, things become more complicated if we work in more complex settings. Sharing and spreading knowledge, maintaining systems and software programs, and many more things work better if

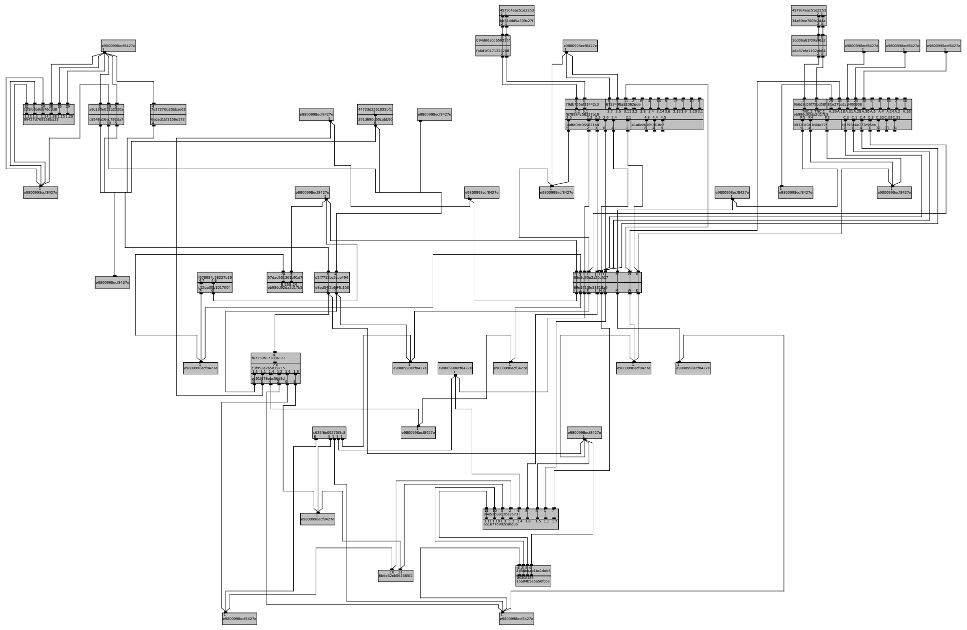


Figure 1.1: An (anonymized) industrial cable plan drawn by our algorithm from Chapter 7.

things are kept simple, light, and clear. We also try to keep geometry simple in this book. We consider only two-dimensional space and mostly the Euclidean norm therein, i.e., the plane as we know it from a sheet of paper or a computer screen. As geometric objects, we mainly use line segments. We can easily describe them by the coordinates of their endpoints, which we can also choose to be integers.[1] When using them for the edges in graph drawings, we get the most simple and widespread type of graph visualizations: *node-link diagrams*.

Finally, we present two practical applications based on content presented in this book. The one is computing an industrial cable plan automatically and the other is simplifying schematic geographic maps when zooming out.

Computing cable plans (see Figure 1.1 for an example) automatically has become necessary in industry because, when working with complex machines where many different combinations and variations of modular components are possible, drawing them all by hand becomes impractical. The phenomenon of ending up with huge numbers of possibilities when combining things is known as *combinatorial explosion* [wik22b]. It often occurs in our world, and typically, it follows some kind of exponential growth. It has been shown multiple times [WS75, CFL19], however, that human beings tend to strongly underestimate exponential growth. The idea of generating cable plans can be applied to multiple different types of industrial

[1] However, in order not to restrict ourselves by the choices of the computational model, we allow computing with real numbers and real-numbered coordinates.

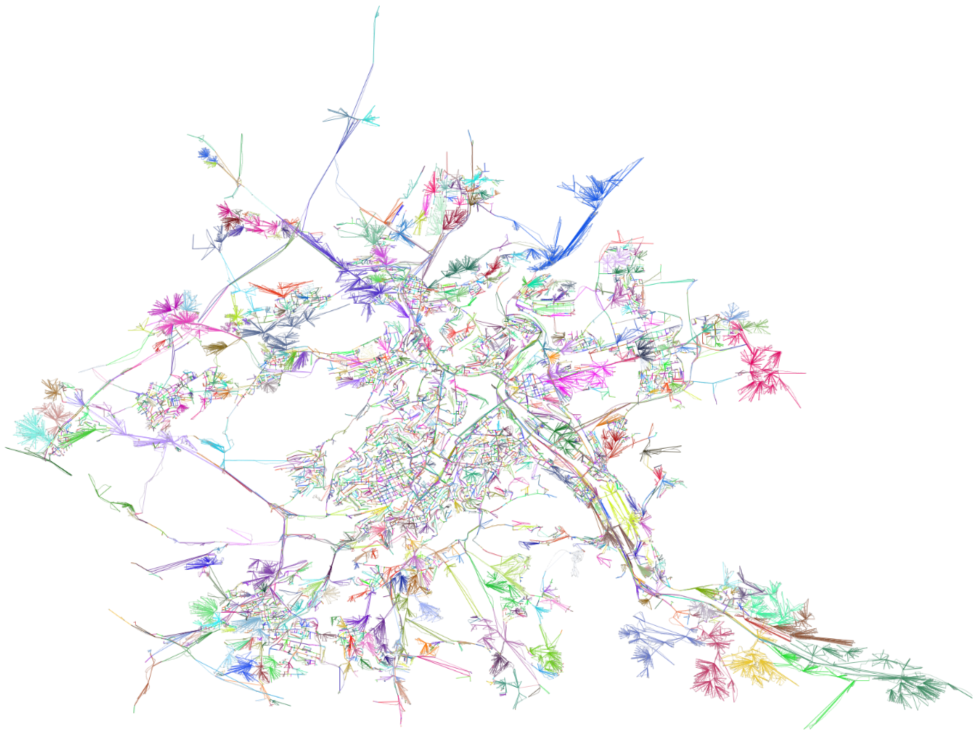


Figure 1.2: Original and simplified map of the road network of Stuttgart taken from Bosch et al. [BSS+21] whose approach is based on the concept of *polyline bundles* introduced in Chapter 9.

networks and beyond, among them local area network plans, circuit plans, networks of product processing chains, and metabolic graphs.

Simplifying schematic geographic maps (see Figure 1.2 for an example) is important since precise data is available that has to be condensed to fit into limited space, for instance, into a piece of paper of a given size. Automatically computing such simplifications has become (even more) important in the times of computers, smartphones, and the Internet where most of the navigation is done with the help of digital maps. There, users zoom in and out via multiple zoom levels. Manually simplifying every part of the map once is not very useful in the long run since the data is not static: roads are built or closed, cities grow, and even borders and coastlines sometimes change.

Outline of the Book

Next, we give an overview of the content of this book. In Chapter 2, we introduce and define the most important concepts and structures used in the following chapters, and we fix our terminology. Readers who are familiar with the subject matter can safely skip this chapter and come back when needed. The subsequent chapters do not build upon each other. They can be read independently of each other as desired.

Part I: Drawing Graphs – Theoretical Results

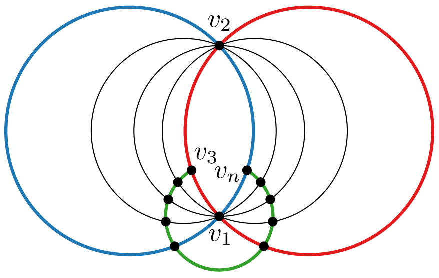
In four chapters of this book, we cover some aspects of graph (drawing) theory whose common feature is the use of straight-line segments in the corresponding drawings. Two of these chapters rely on the concept of *line-segment intersection graphs*: given an arrangement of line segments in the plane, their intersection graph has a vertex for each line segment, and it has an edge between two vertices if and only if the corresponding line segments intersect. In the other two chapters, we consider graph drawings of low *visual complexity*, that is, graph drawings using only few geometric primitives. In our case, these geometric primitives are line segments and distinct slopes of line segments.

Chapter 3: *Recognizing Stick Graphs with and without Length Constraints.*

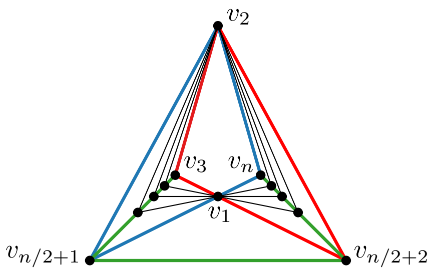
Stick graphs are intersection graphs of horizontal and vertical line segments that touch a line of slope -1 from above. In 2018, stick graphs were introduced by Chaplick et al. [CFHW18] who posed as an open problem the complexity of recognizing stick graphs (Stick). Shortly after in 2019, De Luca et al. [DHK$^+$19] considered this problem when the order of either one of the two sets is given (Stick$_\mathsf{A}$) and when the order of both sets is given (Stick$_\mathsf{AB}$). They showed how to solve Stick$_\mathsf{AB}$ efficiently, namely, in $\mathcal{O}(|A| \cdot |B|)$ time where $|A|$ and $|B|$ are the numbers of vertical and horizontal sticks, respectively. In this chapter, we present an algorithm for Stick$_\mathsf{AB}$ with a faster running time, namely, $\mathcal{O}(|A| + |B| + |E|)$ where $|E|$ is the number of edges. Moreover, we solve Stick$_\mathsf{A}$ in $\mathcal{O}(|A| \cdot |B|)$ time. We remark that recently Rusu [Rus22] has given an algorithm that solves Stick$_\mathsf{A}$ in $\mathcal{O}(|A| + |B| + |E|)$ time. Also, Rusu [Rus23] has shown that Stick is NP-complete.

Further, we consider variants of these problems where the lengths of the sticks are given as input. Including length constraints to the question of recognizing intersection graphs has been suggested by Cabello and Jejčič [CJ17]. We show that these variants of Stick, Stick$_\mathsf{A}$, and Stick$_\mathsf{AB}$ are all NP-complete. However, quite surprisingly, it turns out that we can solve Stick$_\mathsf{AB}$ with fixed stick lengths efficiently by a linear program if there are no isolated vertices. In other words, having isolated vertices makes Stick$_\mathsf{AB}$ with length constraints NP-hard.

This chapter is based on joint work together with Steven Chaplick, Philipp Kindermann, André Löffler, Florian Thiele, Alexander Wolff, and Alexander Zaft [CKL$^+$20].



(a) Drawing with $n/2$ (here, 6) circular arcs.



(b) Drawing with $3n/2$ (here, 18) line segments.

Figure 1.3: A planar graph whose segment number is 3 times its arc number.

Chapter 4: *Lower Bounds on the Segment Number of Some Planar Graph Classes.*

The *segment number* of a planar graph G is the smallest number of line segments needed for any planar straight-line drawing of G. In 2007, Dujmović et al. [DESW07] introduced this measure for the visual complexity of planar graphs. They also give optimal algorithms to determine the segment number of trees and worst-case optimal algorithms for maximal outerplanar graphs, 2-trees, and planar 3-trees.

Instead of considering worst-case instances, i.e., the graphs in a graph class that require the maximum number of segments with respect to the number of vertices, we prove the first *universal lower bounds* on the segment number of maximal outerpaths, maximal outerplanar graphs, 2-trees, and planar 3-trees. In other words, we show that any graph of these graph classes requires a specific number of line segments to be drawn crossing-free with straight-line edges. Our lower bounds are linear in the number of edges of the given graph. For maximal outerpaths and planar 3-trees, we show that our bounds are tight up to a small additive constant.

Moreover, we generalize the result on maximal outerpaths to circular arcs (similar to the segment number, Schulz [Sch15] has introduced the *arc number*) and to other arrangements of curves in the plane crossing pairwise at most k times. We call them *pseudo-k-arc* arrangements due to the analogous definition of pseudoline arrangements and pseudocircle arrangements. We also ask the question if and how much using circular arcs instead of line segments can reduce the visual complexity. Therefore, we do some initial investigations on the ratio between the segment number and the arc number of the same graph; see Figure 1.3 for an example.

We also give a simple optimal algorithm for cactus graphs, generalizing the above-mentioned result for trees.

This chapter is based on joint work together with Ina Goeßmann, Jonathan Klawitter, Boris Klemz, Felix Klesen, Stephen G. Kobourov, Myroslav Kryven, and Alexander Wolff [GKK+22].

Chapter 5: *Upward-Planar Drawings with Three and More Slopes.*

The *planar slope number* of a planar graph G is the smallest number of slopes needed for the edges in any planar straight-line drawing of G. Like the segment number, the planar slope number is a measure of the visual complexity of a graph drawing. Besides the planar slope number for undirected graphs, one can define the *upward-planar slope number* for directed graphs, where we additionally require that any edge in a drawing points upwards. Unfortunately, determining the upward and the usual planar slope number of a graph is, in general, hard in the existential theory of the reals ($\exists\mathbb{R}$) [Hof17, Qua21].

Instead of asking "how many slopes are necessary or sufficient to draw a given graph", we turn the question around and ask "given a set of k slopes, which graphs can we draw?" Klawitter and Mchedlidze [KM22] have investigated this question for $k = 2$.

In this chapter, we study this question for any constant number of slopes with a special focus on the case of three slopes. We show that deciding whether a given directed graph admits an upward-planar drawing with k slopes is NP-hard for outerpaths when $k = 3$ (which implies outerplanar graphs) and for planar graphs when $k \geq 3$. On the positive side, we can decide whether a given directed inner triangulation admits an upward-planar 3-slope drawing, and we can decide whether a given directed cactus graph admits an upward-planar drawing with k slopes. In the affirmative, we can construct such a drawing with a running time that is fixed-parameter tractable (FPT) with respect to k. Furthermore, in linear time, we can determine the minimum number of slopes required for a given tree and compute the corresponding drawing.

This chapter is based on joint work together with Jonathan Klawitter [KZ23]. Moreover, it incorporates ideas from the bachelor thesis of Joshua Geis [Gei22], which I have supervised.

Chapter 6: *Coloring Mixed and Directional Interval Graphs.*

Above, we have already introduced colorings of graphs. We now generalize this concept to mixed graphs. A *mixed graph* is a graph that has, besides a set of undirected edges, also a set of directed arcs such that, between any pair of vertices, there is at most one edge or arc. A *proper coloring* of a mixed graph G is a function c that assigns to each vertex in G a positive integer such that, for each edge uv in G, $c(u) \neq c(v)$ and, for each arc (u, v) in G, $c(u) < c(v)$. Clearly, to admit a proper coloring, it is a necessary and a sufficient condition that a mixed graph does not contain a directed cycle. For a cycle-free mixed graph G, the *chromatic number* $\chi(G)$ is the smallest number of colors in any proper coloring of G. Without restrictions on the graphs, determining the chromatic number of mixed graphs is NP-hard because it is an immediate generalization of coloring undirected graphs.

In this chapter, we consider interval graphs. An *interval graph* is the intersection graph of line segments in one dimension, i.e., intervals on the

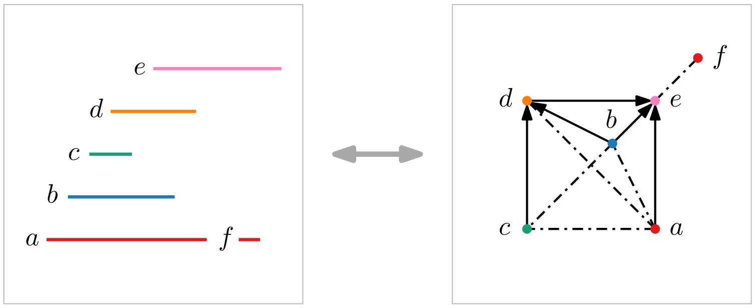


Figure 1.4: Example of an interval representation (left) and the corresponding directional interval graph (right). Intervals are the vertices, and there is an undirected edge (dash dotted) if intervals nest and an arc towards the right interval if the intervals overlap. Here, the chromatic number of the directional interval graph is five.

real line. For an interval graph, we call a corresponding set of intervals an *interval representation*. We show that it is NP-hard to compute the chromatic number of a mixed interval graph. This is in stark contrast to the well-known results that the chromatic number of (undirected) interval graphs [Gol80] and directed acyclic graphs [HKdW97] can be computed in linear time.

A *directional interval graph* (see Figure 1.4) is a mixed interval graph that has an interval representation with the following properties. Such a graph has an (undirected) edge between every two intervals where one is contained in the other and an arc between every two overlapping intervals, directed towards the interval that starts and ends to the right of the other one. We show that we can determine, for a directional interval graph G, its chromatic number $\chi(G)$ and find an optimal coloring of G in $\mathcal{O}(n \log n)$ time, where n is the number of vertices of G.

We also consider *bidirectional interval graphs*. For such a graph, there exists an interval representation with two types of intervals, which we call *left-going* and *right-going*. For left-going intervals, the edges and arcs are defined as in directional interval graphs. For right-going intervals, the symmetric definition applies, that is, an arc is directed towards the interval that starts and ends to the left of the other one. Moreover, there is an edge for every pair of a left-going and a right-going interval that intersect. The coloring algorithm for directional interval graphs yields a 2-approximation of the chromatic number of bidirectional interval graphs.

Coloring bidirectional interval graphs has applications in routing edges in layered orthogonal graph drawing according to the Sugiyama framework. The colors correspond to the tracks for routing the edges. We use our 2-approximation to route the edges of cable plans in Part II.

This chapter is based on joint work together with Grzegorz Gutowski, Florian Mittelstädt, Ignaz Rutter, Joachim Spoerhase, and Alexander Wolff [GMR+22].

Part II: Drawing Graphs – Applied Results

In the second part, we investigate graph drawing algorithms employed in practical applications. It consists of one chapter that concerns cable plans. We think, however, that the content of this chapter is applicable to many domains where a similar drawing style is used. Again, our drawing style is line-segment based: we draw a vertex as an axis-aligned rectangle and we draw an edge as a series of vertical and horizontal line segments.

Chapter 7: *Layered Drawing of Undirected Graphs with Port Constraints.*

In this chapter, we investigate a practical method to draw cable plans of complex machines. Such plans consist of electronic components and cables connecting specific ports of the components. Since the machines are configured for each client individually, resulting in many different combinations, cable plans need to be drawn automatically. The drawings must be well readable so that technicians can use them to debug the machines. In order to model plug sockets, we introduce *port groups*: within a group, ports can change their position (which we use to improve the aesthetics of the layout), but together the ports of a group must form a contiguous block. Moreover, if two plugs or sockets are attached to each other, pairs of corresponding ports need to be drawn on the same vertical or horizontal line. We call such configurations *port pairings*.

We approach the problem of drawing such cable plans by extending the well-known Sugiyama framework [STT81] such that it incorporates ports, port groups, and port pairings. Since the framework assumes directed graphs, we propose two ways to orient the edges of the given undirected graph: orienting the edges along a breadth-first search tree, and computing a force-directed graph layout where we orient the drawn edges upwards. For an example of a cable plan drawn by our algorithm see Figure 1.1.

We compare these methods experimentally, both on real-world data and synthetic data that carefully simulates real-world data. We describe in detail how we generate the synthetic cable plans, which we call *pseudo cable plans*. We measure the aesthetics of the resulting drawings by counting bends and crossings. Using these metrics, we experimentally compare our approach to *Kieler* [SSvH14], a similar library for drawing graphs under the Sugiyama framework in the presence of port constraints. Kieler, however, does not allow the user to specify port groups and port pairings. Our method produced 10–30 % fewer edge crossings, while performing equally well or slightly worse than Kieler with respect to the number of bends and the time used to compute a drawing.

This chapter is based on joint work together with Julian Walter, Joachim Baumeister, and Alexander Wolff [ZWBW22].

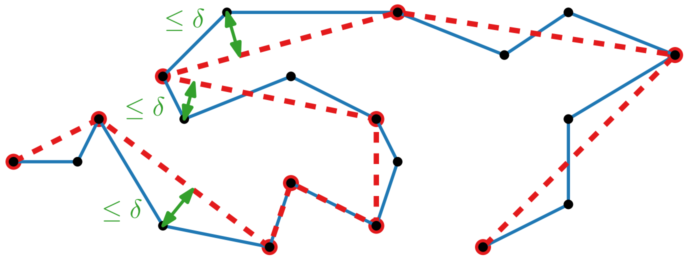


Figure 1.5: A polyline (solid blue) and a simplification of that polyline (dashed red).

Part III: Simplifying Polylines

For the third and last part of this book, we leave the area of drawing graphs. Instead, we consider polylines. A *polyline* is a sequence of line segments defined by a sequence of points, which we call *vertices*. We study the problem of abstracting such a polyline by removing some inner vertices. When vertices are removed, the resulting polyline needs to remain sufficiently similar to the original polyline.

Chapter 8: *Faster Polyline Simplification under the Local Fréchet Distance.*

Given a polyline, the polyline simplification problem asks for a minimum-size subsequence of the vertices defining a new polyline whose distance to the original polyline is at most a given threshold under some distance measure, usually the local Hausdorff or the local Fréchet distance; see Figure 1.5. Here, *local* means that, for each line segment of the simplified polyline, only the distance to the corresponding sub-curve in the original polyline is measured.

This minimization problem is polynomial-time solvable (for the standard distance measures) as first shown by Imai and Iri [II88] for the local Hausdorff distance via a cubic-time algorithm. Later, Chan and Chin [CC96] improved this to quadratic running time. For the local Fréchet distance, the situation has been more intricate. Doubtlessly, cubic running time is possible as shown by Godau [God91] adjusting the Imai–Iri algorithm to the Fréchet metric. This running time has been cited as state-of-the-art many times for the last 30 years even in influential articles. Very recently Buchin et al. [BvdHO$^+$22] showed how to improve this running time to $\mathcal{O}(n^{5/2+\varepsilon})$ where n is the number of vertices of the polyline for any $\varepsilon > 0$ as an application of a sophisticated data structure to store polylines for Fréchet distance queries.

There are, however, older techniques and concepts available in the literature that are capable of solving this problem in almost-quadratic time. Namely, Melkman and O'Rourke [MO88] introduced a geometric data structure to solve polyline simplification under the local Hausdorff distance in $\mathcal{O}(n^2 \log n)$ time, and Guibas et al. [GHMS93] considered polyline simplification under the Fréchet distance as *ordered stabbing* and provided an algorithm with a running time of $\mathcal{O}(n^2 \log^2 n)$, but they did not restrict the simplified polyline to use only vertices of the original polyline.

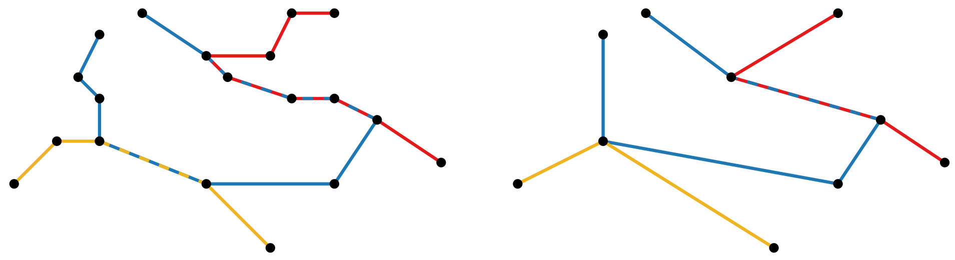

(a) before consistent simplification (b) after consistent simplification

Figure 1.6: Consistent simplification of a polyline bundle with three polylines.

We show that their techniques can be adjusted to solve polyline simplification under the local Fréchet distance in $\mathcal{O}(n^2 \log n)$ time. This algorithm may serve as a more efficient subroutine for multiple other algorithms. We provide a simple algorithm description as well as rigorous proofs to substantiate our claims. We also investigate the geometric data structure introduced by Melkman and O'Rourke, which we refer to as *wavefront*, in more detail and point out some of its interesting properties. As a result, we can prove that under the L_1- and L_∞-norms, the algorithm can be significantly simplified and then only requires a running time in $\mathcal{O}(n^2)$. We also define a natural class of polylines where our algorithm always achieves this running time in the L_2-norm (i.e., Euclidean norm), too.

This chapter is based on joint work together with Peter Schäfer and Sabine Storandt [SSZ23].

Chapter 9: *Consistent Simplification of Polyline Bundles:*

In this chapter, we propose and study a generalization of the polyline simplification problem. Instead of a single polyline, we are given a *bundle* of polylines, that is, a set of polylines possibly sharing some line segments and vertices. The task is to simplify each polyline L of a given polyline bundle by keeping a subset of its vertices such that (i) the local Hausdorff or Fréchet distance between L and its simplified counterpart does not exceed a given distance threshold δ, (ii) a shared vertex is either kept or discarded in all polylines of the polyline bundle (we refer to this requirement as *consistency*) and (iii) the number of kept vertices in the polyline bundle is minimized. See Figure 1.6 for an illustration of this problem. To justify the problem definition, we argue that consistency is crucial to get meaningful and aesthetically pleasing outputs.

Regarding the computational complexity of this problem, we prove, for both distance measures, that polyline bundle simplification is NP-hard to approximate within a factor of $n^{1/3-\varepsilon}$ for any $\varepsilon > 0$ where n is the number of vertices in the polyline bundle. This inapproximability even applies to planar inputs and also to instances with only two polylines.

However, we identify the sensitivity of the solution to the choice of the distance bound δ as a reason for this strong inapproximability. In particular, we prove that if we employ the local Fréchet distance and allow δ to be exceeded by a factor of 2 in the solution, then we can find a simplified polyline bundle with no more than $\mathcal{O}(\log(\ell + n)) \cdot \mathsf{OPT}$ vertices in polynomial time, where ℓ is the number of polylines in the bundle and OPT is the number of vertices in a minimum-size simplification of the bundle with respect to δ. Such an approximation algorithm where we can violate a constraint of the input by a specific factor is known as *bi-criteria approximation algorithm.*

In addition, we show that finding a minimum-size simplification is fixed-parameter tractable (FPT) in the number of vertices being shared between different polylines of the bundle.

This chapter is based on joint work together with Joachim Spoerhase and Sabine Storandt [SSZ20]. The strengthened hardness proof for planar inputs stems from joint work together with Yannick Bosch, Peter Schäfer, Joachim Spoerhase, and Sabine Storandt [BSS$^+$21].

Following these seven chapters presenting research results, this book ends with a conclusion containing an extensive list of open problems in Chapter 10.

Chapter 2
Preliminaries

In this chapter, we briefly introduce important structures and concepts and fix our notation. The definitions used here were chosen to provide consistency throughout the following chapters and to adhere to established standards. Readers that are familiar with algorithms, complexity, graphs, and geometric concepts may well skip this chapter and come back when needed. For a more extended introduction, we refer to textbooks about these topics – for instance, *Introduction to Algorithms* by Cormen, Leiserson, Rivest, and Stein [CLRS22] or the *Handbook of Graph Drawing and Visualization* edited by Tamassia [Tam13].

2.1 Algorithms and Complexity

An *algorithm* is a sequence of elementary instructions, which can be executed by a machine that given some input, performs some computations on it and may give some output. If the sequence of instructions is the same every time an algorithm is executed with the same input, we say the algorithm is *deterministic*; otherwise it is *non-deterministic*. Most notably, the non-deterministic algorithms that include random input or random decisions are *randomized* algorithms. Here, we consider only deterministic algorithms.

The *running time* (or *runtime* for short) of an algorithm is a function describing the number of elementary computing operations (e.g., doing an arithmetic calculation or a comparison of two numbers) that are performed when executing the algorithm with respect to the size of the input or some parameters of the input. Usually, we do not specify the exact number of operations. Instead, we specify a class of functions describing the asymptotic growth of the running time. Here, we use the *Landau notation*, also known as *big-O notation*.

Definition 2.1 (Big-O Notation)**.** Let $f, g\colon \mathbb{N} \to \mathbb{R}$ be functions. We say that f is in $\mathcal{O}(g)$ if there are constants $c, n_0 > 0$, such that for all $n \geq n_0$ it holds that $f(n) \geq c \cdot g(n)$.

For instance, the function $f(n) = 7n^2 - 2\log n + 1000$ is in $\mathcal{O}(n^2)$ and also in $\mathcal{O}(n^5)$, but not in $\mathcal{O}(n \log n)$. Hence, "$\mathcal{O}$" describes an upper bound for the asymptotic behavior of a function. Similarly, we define "Ω" as a lower bound, i.e., $\Omega(g)$ is the set of all functions that grow asymptotically at least as fast as g. If we want strict upper and lower bounds, we use lowercase letters instead. So, $o(g)$ is the set of all functions that grow asymptotically strictly less than g, and $\omega(g)$ is the set of all functions that grow asymptotically strictly greater than g. To specify a precise asymptotic behavior, we sometimes use "Θ": a function f is in $\Theta(g)$ if f is in both $\mathcal{O}(g)$ and $\Omega(g)$.

Complexity Classes and Reductions. We use algorithms to solve given problems. However, depending on the problem, any algorithm solving that problem exactly may need at least a certain running time. For example, it has been shown that any algorithm sorting n numbers based on comparisons has a runtime in $\Omega(n \log n)$. There are even harder problems; problems that do not admit an algorithm running in $\mathcal{O}(n^c)$ time where n is the size of the input and c is any constant. We say that this problem cannot be solved in polynomial time.

If there exists a deterministic algorithm solving a problem with a runtime in $\mathcal{O}(n^c)$ for some constant c, i.e., polynomial time, we say that this problem lies in the complexity class P. If there exists a deterministic algorithm verifying a given solution of a problem with a runtime in $\mathcal{O}(n^c)$, we say that this problem lies in the complexity class NP. It is an unresolved problem in computer science – and also one of the seven Millennium Prize Problems of mathematics – whether $\mathsf{P} \subsetneq \mathsf{NP}$ or $\mathsf{P} = \mathsf{NP}$. However, it is most widely assumed that $\mathsf{P} \subsetneq \mathsf{NP}$. Then, there exist problems that are in NP, but not in P – most notably, the "most difficult" problems of the class NP, which we call *NP-complete*.

Given any problem $\mathcal{A}$ and an NP-complete problem $\mathcal{B}$, $\mathcal{A}$ is in NP if there exists a deterministic algorithm with polynomial running time that translates $\mathcal{A}$ into $\mathcal{B}$ and re-translates the corresponding solution of $\mathcal{B}$ to a solution of $\mathcal{A}$. We call such an algorithm a *polynomial-time reduction* and we say that we can *reduce* $\mathcal{A}$ to $\mathcal{B}$. If, on the other hand, we can reduce problem $\mathcal{B}$, which is NP-complete (or harder), to $\mathcal{A}$, this means that $\mathcal{A}$ is at least as hard as the hardest problems in NP. We then say $\mathcal{A}$ is *NP-hard*. Note that this does not mean that $\mathcal{A}$ is NP-complete because $\mathcal{A}$ itself might not lie in the complexity class NP. Only if $\mathcal{A}$ is in NP, then $\mathcal{A}$ is also NP-complete. This notion of *completeness* and *hardness* can be applied to other complexity classes as well.

Quite interestingly, many practically relevant problems are NP-complete. These include graph coloring, scheduling, computing a cheapest round trip visiting a set of cities, partitioning a set of numbers into two subsets of equal sum, or deciding whether a given Boolean formula is satisfiable.

Some geometric problems require computations with real numbers even to verify a solution. These problems are then not contained in NP but might lie in its superclass $\exists\mathbb{R}$, which is called *existential theory of the reals.* The generic $\exists\mathbb{R}$-complete problem is to decide whether there are real numbers $x_1, \ldots, x_m$ that satisfy a given quantifier-free formula involving equalities and inequalities of real polynomials. The complexity class $\exists\mathbb{R}$ is defined as the set of all problems that can be expressed in this form. If we can reduce an $\exists\mathbb{R}$-hard problem to another problem $\mathcal{A}$, then $\mathcal{A}$ is $\exists\mathbb{R}$-hard, too. See the works by Matoušek [Mat14] and Schaefer [Sch09] for more information on the complexity class $\exists\mathbb{R}$ and some problems in this class.

For practical applications and large instances, only algorithms with polynomial running time are applicable – we thus call them *efficient.* To still tackle NP-hard problems, it makes sense to consider a more fine-grained analysis depending on some parameters of the input. This is due to the fact that for a family of instances that have a large size but are small in this parameter, we can still solve the problem efficiently. Given some parameter k (this can also be the solution size), we say that problem $\mathcal{A}$ is *fixed-parameter tractable (FPT)* with respect to k if there is an algorithm solving $\mathcal{A}$ with a runtime in $\mathcal{O}(f(k) \cdot n^c)$ where f is some computable function and c is a constant. If there is an algorithm solving $\mathcal{A}$ with a runtime in $\mathcal{O}(n^{f(k)})$, we say that $\mathcal{A}$ is *slice-wise polynomial (XP)* with respect to k. Given a parameter, we can associate FPT and XP with a set of problems and then it holds that $\mathsf{FPT} \subsetneq \mathsf{XP}$.

Approximations. Another way to deal with an NP-hard problem is to describe an *α-approximation algorithm*, where α is a constant or a function in the input size. An algorithm is called an *α-approximation algorithm* if it has polynomial runnning time and returns a (valid) solution whose value is at most by a factor of α off the value of an optimal solution.

A *bi-criteria approximation* is a generalization of a (classical) approximation where it is allowed to violate a certain constraint by a given factor. In particular, an algorithm is called a *bi-criteria (α, β)-approximation algorithm* if it runs in polynomial time and produces a solution of value at most $\alpha \cdot \mathsf{OPT}$, where OPT is the value of an optimal solution, while relaxing a constraint of the problem by a factor of β.

2.2 Graphs

An (undirected) *graph* G is a tuple (V, E) where V is a set of *vertices* and E is a set of unordered pairs of distinct vertices. We call these pairs of vertices (undirected) *edges* and we denote an edge connecting vertices u and v by uv. The number of vertices and the number of edges of a graph are denoted by n and m, respectively.

A *directed graph* (or *digraph*) is a graph whose edges are ordered pairs of distinct vertices called *directed edges*. For example, the directed edge uv represents a directed relation from u to v and is different from the edge vu (both can exist in a digraph). We say u is *directed* (or alternatively *oriented*) towards v.

Note that in our definition of graphs, we do not allow parallel edges and self loops, that is, edges that connect a vertex to itself. Consequently, a graph with n vertices has at most $(n^2 - n)/2$ edges and a digraph has at most $n^2 - n$ edges.

A *mixed graph* is a triple (V, E, A), where E is a set of undirected edges and A is a set of directed edges. For us, a mixed graph has at most one edge between each pair of vertices – either an undirected edge or one of the two possible directed edges. Hence, the maximum number of edges of a mixed graph is also $(n^2 - n)/2$.

For a digraph or a mixed graph G, we call the graph obtained by ignoring edge directions and then joining parallel edges the *underlying undirected graph of* G, which we denote by $U(G)$.

For a more compact notation – this is in particular helpful when we deal with mixed graphs – we often use the term *arc* instead of directed edge, and we mean an undirected edge when we use just *edge*. To also distinguish them in mathematical notation, we use, instead of uv, $\{u, v\}$ and (u, v) for an undirected edge between u and v and for an arc from u to v, respectively.

Neighborhood and Vertex Degrees. Given an undirected graph, we say that two vertices u and v are *neighbors* or *adjacent* if there is an edge uv. The edge uv is *incident* to the vertices u and v and, the other way around, the vertices u and v are *incident* to the edge uv as well. Two edges are *adjacent* if they share a common vertex. The (open) neighborhood $N(v)$ of a vertex v is the set $\{u \in V : uv \in E\}$, and the (closed) neighborhood $N[v]$ is the set $N(v) \cup \{v\}$.

The *degree* $\deg(v)$ of a vertex v is the size of its open neighborhood. Clearly, the sum of the degrees of all vertices of a graph is twice the number of edges of the same graph. If we have a digraph $G = (V, E)$, then we distinguish, for a vertex v, between *indegree* $\deg_{\text{in}}(v)$ and *outdegree* $\deg_{\text{out}}(v)$ of v, where $\deg_{\text{in}}(v) = |\{uv : uv \in E\}|$ and $\deg_{\text{out}}(v) = |\{vw : vw \in E\}|$.

If we can separate the vertex set of a graph G into two disjoint sets A and B such that all neighbors of a vertex in A lie in B and vice versa, we say G is *bipartite*.

Sometimes, we consider graphs where every vertex has the same degree k. We call such a graph a *k-regular graph*. For $k = 3$, such a graph is called *cubic*. Clearly, a 0-regular graph is a set of isolated vertices, a 1-regular graph is a set of disjoint edges, and a 2-regular graph is a set of disjoint cycles.

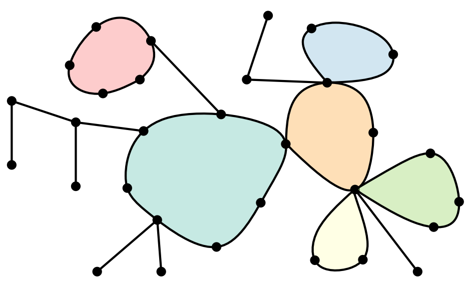
Subgraphs. A graph H is called a *subgraph* of a graph $G = (V, E)$ if we can obtain H from G by removing a set of vertices and edges. Moreover, a subgraph H of G is called *induced* if we can obtain H from G by removing a set of vertices, i.e., we keep exactly those edges none of whose endpoints is removed. For a set of vertices U, $G[U]$ is the induced subgraph of G obtained by removing the vertices in $V \setminus U$ from G.

Paths and Cycles. A *path* P is a sequence $\langle v_1, v_2, \dots, v_\ell \rangle$ of $\ell \geq 2$ distinct vertices such that for each pair $\langle v_i, v_{i+1} \rangle$ $(i \in \{1, \dots, \ell - 1\})$ of consecutive vertices, there is an edge $v_i v_{i+1}$. The *length* of P is $\ell - 1$. In a digraph, we call P a *directed path* if each of these edges is directed from v_i to v_{i+1}.

A *cycle* C is a sequence $\langle v_1, v_2, \dots, v_\ell \rangle$ of $\ell \geq 3$ vertices such that for each pair $\langle v_i, v_{i+1} \rangle$ $(i \in \{1, \dots, \ell - 1\})$ of consecutive vertices there is an edge $v_i v_{i+1}$, and there is the edge $v_\ell v_1$. The *length* of C is ℓ. A cycle C is *simple* if no vertex appears more than once in C. In a digraph, we call $C = \langle v_1, v_2, \dots, v_\ell \rangle$ a *directed cycle* if each of the edges of the cycle is directed from v_i to v_{i+1} and the last edge is directed from v_ℓ to v_1. The graph that consists only of a simple cycle of k vertices is denoted by C_k.

Connectivity and Breadth-First Search. If there is a path P from a vertex v to a vertex u, we say u and v are *connected* (via P). A maximal set of connected vertices is called a *connected component*. If all vertices of a graph lie in the same connected component, we call the graph *connected*.

We can explore a graph and find the connected components of a graph, e.g., by a *breadth-first search (BFS)*. In a BFS, we initialize the graph by coloring each vertex white and we maintain a queue Q where we put the vertices that we explore next. Then, we start at each vertex v and if it is colored white, we put v into the queue Q. While Q is not empty, we extract the first vertex v of Q and consider each vertex


(a) A cactus graph G.

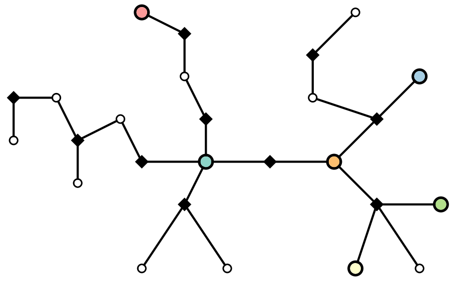

(b) Block-cut tree of G. Block nodes have disk shapes and cut vertex nodes diamond shapes.

Figure 2.1: A cactus and its block-cut tree. Matching cycles and block nodes have the same color.

$u \in N[v]$. If u is colored white, we color u gray, save v as u's predecessor and put u into the queue Q. In the end, when following the predecessor pointers, they induce a tree called *BFS-tree* for each connected component. (A tree is defined next.) Clearly, the runtime of a BFS is $\mathcal{O}(n + m)$ since each vertex is put into the queue once and colored gray, and each edge is explored once from each of its two sides.

We say that a connected component of a graph (or a connected graph) is *biconnected* if we need to remove at least two vertices until the connected component breaks apart into more than one connected component. If a connected graph is not biconnected, we call the vertices whose removal disconnects the graph *cut vertices*. We can generalize the idea of biconnectivity to an arbitrary number k: a connected component is *k-connected* if we need to remove at least k vertices until the connected component breaks apart into more than one connected component. For $k = 3$, we also use the term *triconnected*.

Trees, Cacti, and DAGs. A graph without cycles is a *forest*, and each connected component of a forest is a *tree*. Clearly, a graph (with n vertices) that is a tree has exactly $n - 1$ edges. We call a digraph whose underlying undirected graph is a tree a *directed tree*.[2] If in a directed tree T all but one vertex r have indegree 1 and r has indegree 0, we call T a *rooted tree* and r the *root* of T.

A graph G is called a *cactus* if any two cycles of G share at most one vertex; see Figure 2.1a. The *block-cut tree* of a cactus G has a node for each biconnected component (called *block*) of G, and a node for each cut vertex of G. Moreover, it has an edge between a block B and a cut vertex c if c is part of B; see Figure 2.1b. The block-cut tree of G can be computed in linear time [Tar72]. Note that, for a cactus, each block is either a cycle or an edge. Thus, we distinguish between *cycle blocks* and *edge blocks*. A digraph is a cactus if its underlying undirected graph is a cactus.

A digraph that does not contain a directed cycle is a *directed acyclic graph (DAG)*. Note that the underlying undirected graph of a DAG can contain (undirected) cycles.

[2] In literature, a directed tree is sometimes also called a *polytree*.

Cliques and Complete Graphs. A set of k vertices where each pair of vertices is connected by an edge is a *clique* of size k. The graph that consists only of a clique of size k is denoted by K_k. We call such a graph a *complete graph.*

k-Trees. The *k-trees* are defined inductively. An n-vertex graph G is a *k-tree* if it admits a *stacking order* $\langle v_1, v_2, \dots, v_n \rangle$ of its vertices together with a sequence of induced subgraphs $G_{k+1} = G[\{v_1, v_2, \dots, v_{k+1}\}]$, $G_{k+2} = G[\{v_1, v_2, \dots, v_{k+2}\}]$, $\dots$, $G_n = G$ such that (i) G_{k+1} is a clique on the vertex set $\{v_1, v_2, \dots, v_{k+1}\}$, and (ii) for $i \in \{k+2, \dots, n\}$, the graph G_i is obtained from G_{i-1} by making v_i adjacent to all vertices of a k-clique in G_{i-1}. The vertex addition in step (ii) is called *stacking operation.*

In Section 2.3, we describe some properties of drawings of 2-trees and 3-trees.

Intersection Graphs. We can define a family of graphs by considering arrangements of geometric objects. More precisely, when we are given a collection $\mathcal{S}$ of geometric objects, the *intersection graph of* $\mathcal{S}$ has $\mathcal{S}$ as its vertex set and it has, for each pair of geometric objects $S, S' \in \mathcal{S}$, an edge whenever $S \cap S' \neq \emptyset$. Typical examples are intersection graphs of intervals of the real line, unit disks in the plane, or cuboids in higher dimensions. The set of objects $\mathcal{S}$ is called a *representation* – for instance, an interval representation (for an interval graph) if $\mathcal{S}$ is a set of intervals. A typical question concerning intersection graphs is: *Given a graph G, what is the complexity to decide whether G is an intersection graph for a given type of geometric objects?*

In this book, we consider only one- or two-dimensional objects – namely, interval graphs and stick graphs, where the geometric objects of the former are one-dimensional intervals and the geometric objects of the latter are horizontal or vertical line segments being grounded with their left/bottom endpoint on a line of slope -1. We call the respective intersection graphs *interval graphs* and *stick graphs.* We assume that our intervals and sticks have (i) real numbers as coordinates for the start and endpoints, (ii) they are all closed, i.e., the start and endpoints belong to the object, and (iii) they are non-degenerate, i.e., they have a length greater than 0.

2.3 Graph Drawing

A *drawing* of a graph $G = (V, E)$ is a mapping Γ of G such that $\Gamma \colon V \to \mathbb{R}^2$ and, for $u, v \in V$, $\Gamma(u) = \Gamma(v)$ implies $u = v$, i.e., Γ is injective. In other words, each vertex is mapped to a unique point in the Cartesian plane. Moreover, each edge $e \in E$ is mapped to a simple open Jordan curve $\Gamma_{uv} \colon [0, 1] \to \mathbb{R}^2$ such that $\Gamma_{uv}(0) = \Gamma(u)$ and $\Gamma_{uv}(1) = \Gamma(v)$. For simplicity, we call the points representing vertices also vertices and the Jordan curves representing edges also edges.

Drawings with (Straight-)Line Segments. A drawing of a graph is called *straight-line* if each of its edges is drawn as a straight-line segment; see Figure 2.2.

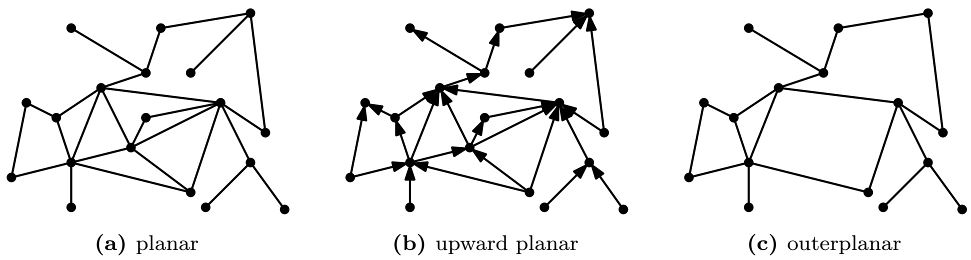

(a) planar **(b)** upward planar **(c)** outerplanar

Figure 2.2: Examples of specific straight-line graph drawings.

If each edge is represented by a sequence of line segments, we call the drawing a *polyline drawing* and the junction points are called *bends*.

Planar Graphs. In graph drawing, the class of graphs that probably received the most attention – aside from trees – are the planar graphs. A drawing of a graph is called *planar* if no pair of edges intersect each other except for the common endpoint of adjacent edges; see Figure 2.2a. A graph is *planar* if it admits a planar drawing´.

In a planar drawing, we call the topologically connected regions of the plane bounded by the edges *faces*. The unbounded face is called the *outer face*. All other faces are *inner faces*. There is a famous formula by Euler that describes the relationship between the number of vertices, edges, faces, and connected components (abbreviated by n, m, f, k, respectively) in a planar graph:

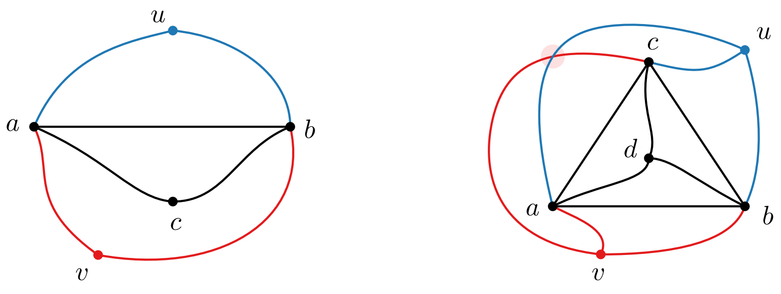
$$n - m + f = k + 1\,.$$

It directly follows that every planar graph has at most $3n - 6$ edges since each face is bounded by at least three edges and each edge bounds at most two faces.

Upward-Planar Graphs. When we consider digraphs, we can extend our notion of planarity by including edge directions. A planar drawing of a digraph where every edge uv is drawn as a monotonic upward curve from u to v is an *upward-planar drawing*; see Figure 2.2b. A digraph that admits an upward-planar drawing is an *upward-planar (di)graph*. Clearly, a necessary condition for a digraph to be upward planar is that it is a DAG.

Outerplanar Graphs. A planar drawing of a graph where all vertices lie on the outer face is called *outerplanar*; see Figure 2.2c. A graph admitting an outerplanar drawing is an *outerplanar* graph. Again, we can extend this notion to digraphs: an outerplanar drawing of a digraph is *upward outerplanar* if all directed edges are drawn such that they point upwards, and a digraph is *upward outerplanar* if it admits an upward-outerplanar drawing.

2-Trees, 3-Trees, and Maximal Outerplanar Graphs. In this book, we also investigate planar straight-line drawings of 2-trees and 3-trees. Recall the inductive



(a) non-outerplanar 2-tree **(b)** non-planar 3-tree

Figure 2.3: A 2-tree and a 3-tree that are not outerplanar and planar, respectively.

definition of k-trees. In the inductive step for 2-trees, observe that we add a vertex to both endpoints of an existing edge. Since we start with K_3, which can be drawn planar, the operation of adding a vertex and two edges can be made while keeping the graph planar. Therefore, all 2-trees are planar graphs.

This is not true for 3-trees. For an example of a non-planar 3-tree, see Figure 2.3b. We start with a planar drawing of K_4 with outer cycle $\langle a, b, c \rangle$. We can add a vertex u being adjacent to $\langle a, b, c \rangle$ while keeping the graph planar. However, when we add another vertex v to $\langle a, b, c \rangle$, we clearly cannot avoid an edge crossing (highlighted in red). Hence, we speak of *planar 3-trees* when mean the set of all 3-trees that are planar.

There is an interesting relationship between outerplanar graphs and 2-trees. Namely, the maximal outerplanar graphs are a subset of the 2-trees. A *maximal outerplanar graph* is an outerplanar graph that admits a drawing where all inner faces are triangles. We can easily see that a maximal outerplanar graph is a 2-tree by incrementally constructing an outerplanar drawing whose inner faces are all triangles. We start with a single triangular face and append the neighboring faces by adding the missing vertex to the endpoints of an existing edge. The reverse is not true. In other words, there are 2-trees that are not maximal outerplanar graphs; see Figure 2.3a. We start with a cycle $\langle a, b, c \rangle$. Now if we stack two vertices u and v onto the edge ab, either c, u, or v lies in an inner face.

Dual Graphs, Outerpaths, and Inner Triangulations. Given a planar drawing Γ, the *dual graph* of Γ has the faces of Γ as its vertex set and it has an edge for each pair of faces that share an edge. The *weak dual graph* of Γ is the dual graph of Γ without the outer face.

Observe that for an outerplanar drawing, the weak dual graph is a tree. A drawing whose weak dual graph is a path is an *outerpath drawing*. A graph is an *outerpath* if it admits an outerpath drawing. Again, we define an *upward-outerpath* drawing (graph) with respect to an outerpath drawing (graph) similarly as an upward-outerplanar drawing (graph) with respect to an outerplanar drawing (graph).

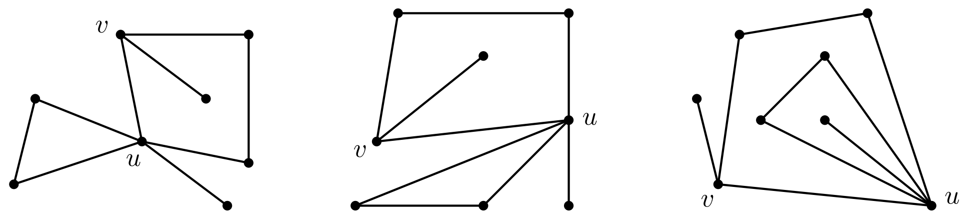


Figure 2.4: Three planar drawings of the same graph. The two left drawings are equivalent in the sense that they belong to the same embedding, while the rightmost drawing belongs to a different embedding. The rotation system differs for the clockwise order of edges around the vertices u and v.

In this book, we additionally require an outerplanar graph and an outerpath to have at least three vertices and to be biconnected, i.e., they do not have “dangling” leaves or paths. (Allowing them can be handled by additional pre- and post-processing steps in Chapters 4 and 5, which we want to avoid there because we do not consider these extra steps to be very interesting or relevant.)

Similar to maximal outerplanar graphs, a *maximal outerpath* is an outerpath that admits an outerplanar drawing where all inner faces are triangles. More generally, we call a graph drawing where all inner faces are triangles an *inner triangulation.*

Embeddings. Consider a planar graph. There is an infinite number of drawings of the same graph. However, some of them are similar in the way the edges are arranged around the vertices; see Figure 2.4. In a planar drawing Γ, we call the clockwise order of edges around each vertex the *rotation system* of Γ. An equivalence class of drawings of a graph G having the same rotation system and the same outer face is a (planar) *embedding* of G. Clearly, two drawings of the same embedding have the same set of faces in a combinatorial sense, that means, for each face of the first drawing, there is a face in the second drawing having the same order of edges and vertices when traversing the boundary of the faces clockwise. The concept of planar embeddings naturally extends to upward-planar and outerplanar embeddings.

An embedding can be seen as a combinatorial description of a drawing. Often we are given a graph G and an embedding of G and we want an algorithm to draw G according to the given embedding. We call a planar graph together with a planar embedding a *plane graph.* Similarly, we call an upward-planar digraph together with an upward-planar embedding and an outerplanar graph together with an outerplanar embedding *upward-plane digraph* and *outerplane graph*, respectively. Since drawings of trees have only one face, only the rotation system matters. Hence, we speak of an *ordered tree* if we are given a tree with a rotation system and we speak of an *unordered tree* otherwise. When we consider a problem where a graph is part of the input, we also distinguish between the *fixed-embedding scenario*, where we are also given an embedding of that graph and this embedding must be preserved, and the *variable-embedding scenario*, where we are given only the graph.

Area of a Graph Drawing. If all vertices of an n-vertex graph drawing Γ are placed on grid points of a regular grid of size $W \times H$, we say the *drawing area* of Γ is $W \times H$. Typically, we give upper bounds such as $\mathcal{O}(n) \times \mathcal{O}(n)$ or $\mathcal{O}(n^2)$ if the ratio of width and height does not matter. If there is no underlying grid of Γ having polynomial size, we say that Γ uses *superpolynomial drawing area.* E.g., this can happen if we use real coordinates or if our vertex coordinates have superlogarithmic length. We say a planar graph (upward-planar digraph) G *requires a drawing area of size* $W \times H$ if there is no planar (upward-planar) drawing Γ of G using less than $W \times H$ drawing area.

Properties of Planar and Upward-Planar Graphs. Testing whether a given graph is planar can be done in linear time as shown by Hopcraft and Tarjan [HT74] in 1974. A classic result in graph drawing is that all planar graphs can be drawn with straight-line edges. This is known as *Fáry's theorem* and has been proven in 1936 by Wagner [Wag36], in 1948 by Fáry [Fár48], and in 1951 by Stein [Ste51]. Moreover, a planar straight-line drawing with quadratic drawing area can be computed in linear time [dFPP90, Sch90].

Similar to Fáry's theorem, Di Battista and Tamassia [DT88] have shown that if a digraph is upward planar, then it also admits an upward-planar straight-line drawing. However, on the algorithmic side, some problems that can be solved efficiently for planar graphs become hard for upward-planar graphs. Garg and Tamassia [GT01] have shown that upward-planarity testing is NP-complete – this remains true even for digraphs with bounded maximum degree Δ, where $\Delta \geq 2$ [KM22]. On the positive side, if an embedding of a digraph is given, upward planarity can be tested in polynomial time as shown by Bertolazzi, Di Battista, Mannino, and Tamassia [BDLM94].

2.4 Polylines

A *polyline* is a series of line segments that are defined by a sequence of d-dimensional points $L = \langle p_1, p_2, \ldots, p_n \rangle$, which we call *vertices.* Unless stated differently, we assume $d = 2$ in this book. By n, we denote the *length* of a polyline. For $1 \leq i \leq j \leq n$, we let $L[p_i, p_j] = \langle p_i, p_{i+1}, \ldots, p_{j-1}, p_j \rangle$, that is, the subpolyline of L starting at vertex p_i, ending at vertex p_j, and including all vertices in between in order. The continuous (but not smooth) curve induced by the vertices of a polyline L is denoted as $c_L \colon [1, n] \to \mathbb{R}^d$ with $c_L \colon x \mapsto (1-x')p_{\lfloor x \rfloor} + x' p_{\lceil x \rceil}$, where $x' = x - \lfloor x \rfloor$. Alternative names for polylines include *polygonal chains*, *polygonal curves*, *broken lines*, and *linestrings*.

Polyline Simplification. Often, we do not need the full detail of a polyline, but prefer (for reasons of space efficiency) a simplified – but still sufficiently similar – version of that polyline. For an illustration, see Figure 2.5. The polyline simplification problem is defined as follows.

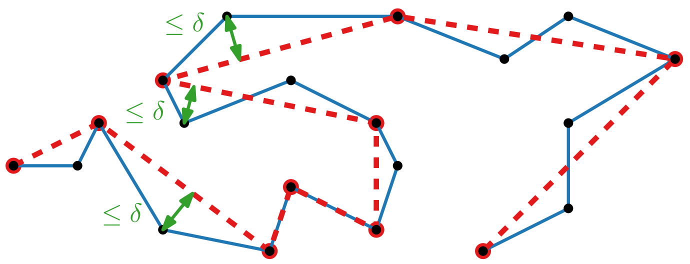


Figure 2.5: A polyline (solid blue) and a simplification of that polyline (dashed red).

Definition 2.2 (Polyline Simplification)**.** Given a polyline $L = \langle p_1, p_2, \ldots, p_n \rangle$, a distance measure d_{X} for determining the distance between two polylines, and a distance threshold parameter δ, the objective is to obtain a minimum-size subsequence S of L such that $p_1, p_n \in S$, and $d_{\mathrm{X}}(L, S) \leq \delta$. We refer to S as a *simplification* of the (original) polyline L.

Note that d_{X} is some distance measure to determine the distance between two polylines as a number. Next, we describe the distance measures usually used in the context of polyline simplification.

Distance Measures. We consider the *Hausdorff* and the *Fréchet* distance in their local and their global variants. We start with the global variants.

Definition 2.3 (Hausdorff Distance)**.** Given two polylines $L = \langle p_1, \ldots, p_n \rangle$ and $L' = \langle q_1, \ldots, q_m \rangle$, the *(undirected) Hausdorff distance* $d_{\mathrm{H}}(L, L')$ is defined as

$$d_{\mathrm{H}}(L, L') = \max \left\{ \sup_{p \in c_L} \inf_{q \in c_{L'}} d(p, q),\ \sup_{q \in c_{L'}} \inf_{p \in c_L} d(p, q) \right\},$$

where sup is the supremum, inf is the infimum and $d(p, q)$ is the distance between the points p and q under some norm (e.g., the Euclidean norm, also known as the L_2-norm; see below).

An often raised criticism concerning the use of the Hausdorff distance is that it measures the similarity of two polylines as planar point sets, but it does not take into account that polylines are an *ordered* sequence of points. In contrast, the Fréchet distance measures the maximum distance between two polylines while traversing them in parallel and is therefore often regarded as the better-suited measure for polyline similarity.

Definition 2.4 (Fréchet Distance)**.** Given two polylines $L = \langle p_1, \ldots, p_n \rangle$ and $L' = \langle q_1, \ldots, q_m \rangle$, the *Fréchet distance* $d_{\mathrm{F}}(L, L')$ is defined as

$$d_{\mathrm{F}}(L, L') = \inf_{\alpha, \beta} \max_{t \in [0,1]} d(c_L(\alpha(t)), c_{L'}(\beta(t))),$$

where $\alpha\colon [0, 1] \to [1, n]$ and $\beta\colon [0, 1] \to [1, m]$ are continuous and non-decreasing functions with $\alpha(0) = \beta(0) = 1$, $\alpha(1) = n$, $\beta(1) = m$.

Traditionally, in the context of polyline simplification, the *local* Hausdorff and the *local* Fréchet distance are used, which measure the Hausdorff or Fréchet distance only between a line segment $\langle p_i, p_j \rangle$ of the simplification and its corresponding subpolyline $L[p_i, p_j]$ in the original polyline. Similar to the advantage of the Fréchet distance over the Hausdorff distance, the advantage of a local over a global distance measure is that they compare only related parts of a polyline and its simplification. Hence, using the local Fréchet distance for polyline simplification as we do in this book arguably is a sensible choice. We define only the local Fréchet distance d_{lF} formally – the local Hausdorff distance d_{lH} is defined similarly.

Definition 2.5 (Local Fréchet Distance). Given a polyline $L = \langle p_1, p_2, \ldots, p_n \rangle$ and a simplification $S = \langle p_1 = p_{s_1}, p_{s_2}, \ldots, p_{s_{|S|}} = p_n \rangle$ of L, the *local Fréchet distance* $d_{\mathrm{lF}}(S, L)$ is defined as

$$d_{\mathrm{lF}}(S, L) = \max_{i \in 1,\ldots|S|-1} d_{\mathrm{F}}(\langle p_{s_i}, p_{s_{i+1}} \rangle, L[p_{s_i}, p_{s_{i+1}}]) ,$$

where $\langle p_{s_i}, p_{s_{i+1}} \rangle$ is the polyline of length two (i.e., the line segment) from p_{s_i} to $p_{s_{i+1}}$ and $L[p_{s_i}, p_{s_{i+1}}]$ is the (sub)polyline obtained by taking the substring from p_{s_i} to $p_{s_{i+1}}$ of L.

Observe that the Hausdorff distance is a lower bound for the Fréchet distance. When considering the Fréchet distance, we may say that the distance threshold δ is respected or exceeded already in the Hausdorff distance since not exceeding δ with respect to the Hausdorff distance is a necessary condition to not exceed δ with respect to the Fréchet distance.

When using the local Fréchet distance, we can tell for each pair of vertices $\langle p_i, p_j \rangle$ (for $1 \le i < j \le n$) in the original polyline independently whether a simplification may contain the line segment $\langle p_i, p_j \rangle$ or not by considering the Fréchet distance only between the line segment $\langle p_i, p_j \rangle$ and its corresponding subpolyline. When considering such a pair $\langle p_i, p_j \rangle$ as a line segment for a simplification, we call it a *shortcut*. If the distance between a segment $\langle p_i, p_j \rangle$ and its corresponding subpolyline does not exceed the distance threshold δ, we call it a *valid* shortcut. Note that trivially $\langle p_i, p_{i+1} \rangle$ is always a valid shortcut for any $i \in \{1, \ldots, n-1\}$. If, in a simplification, $\langle p_i, p_j \rangle$ is a line segments, then the vertices $p_{i+1}, \ldots, p_{j-1}$ do not occur in this simplification. Hence, we say these vertices are *skipped* by the shortcut $\langle p_i, p_j \rangle$.

$\mathbf{L_p}$-Norms. In the definition of the Hausdorff and Fréchet distance, we can choose how the distance between two d-dimensional points, or vectors, is determined. Typically, a vector norm is used for this purpose. For $p \in [1, \infty)$, the *L_p-norm* of a vector $x \in \mathbb{R}^d$ is defined as $\|x\|_p = \left(\sum_{i=1}^{d} |x_i|^p\right)^{1/p}$. For $p = 1$, the L_p-norm is called the *Manhattan* norm; for $p = 2$, it is called the *Euclidean* norm. For $p \to \infty$, the L_∞-norm is called the *maximum* norm and it is defined as $\max_{i=1,\ldots,n} |x_i|$.

The unit sphere S_p^d is the set of points in $\mathbb{R}^d$ within unit distance to the origin. While this unit is conventionally set to 1, we instead use the given distance parameter δ

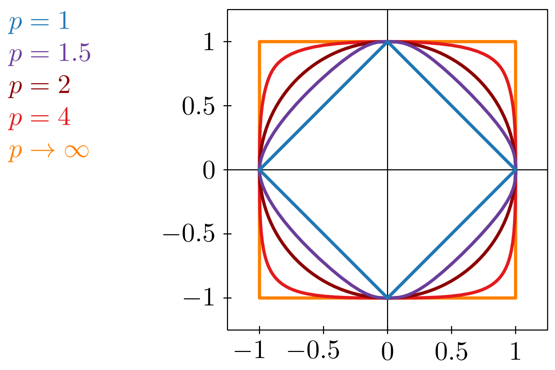


Figure 2.6: Unit disks in L_p-norms for selected values of p. In the context of polyline simplification, we may use a radius of δ, which is the given distance parameter, instead of 1.

as this allows for easier integration to polyline simplification with error bound δ. We hence define $S_p^d = \{x \in \mathbb{R}^d \colon \|x\|_p \leq \delta\}$.

For $d = 2$, S_p^2 is also called the *unit disk* in the L_p-norm. In the L_1- and L_∞-norms, the unit disks actually form squares with side lengths $\sqrt{2}\delta$ and 2δ, respectively. In the L_2-norm, the unit disk is bounded by a circle with radius δ. In the L_p-norm with p between 2 and ∞, the unit disk is bounded by a supercircle which, for larger values of p, resembles more and more a square; see Figure 2.6. A similar statement holds for the L_p-norm with p between 1 and 2. We refer to a contiguous subset of the boundary of a unit disk in any L_p-norm as an *arc*.

Part I

Drawing Graphs – Theoretical Results

Chapter 3
Recognizing Stick Graphs with and without Length Constraints

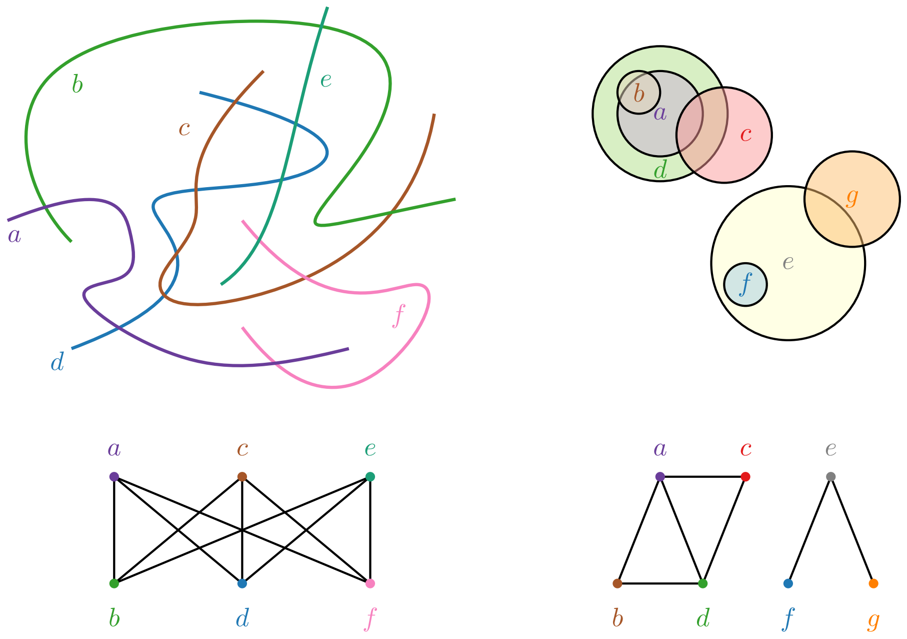
We start the part of theoretical graph drawing results with an instance of the classical problem of recognizing intersection graphs. In some sense, this is more about specific graph classes and is different from usual graph drawing where graphs are represented such that the vertices are points (or small shapes) and the edges are curves in the plane connecting their corresponding vertices. We consider that style of representation in the two succeeding chapters and in the part about applied graph drawing.

3.1 Introduction

Already in 1945 Marczewski [Szp45] showed that any graph may be represented as an intersection graph of sets. This problem was also considered by Čulík [Čul64] in 1964, and by Erdős and Pósa [EGP66] in 1966. However, this statement is not true if we consider connected geometric objects in the plane instead of (more general) sets. For a given collection $\mathcal{S}$ of connected geometric objects, the *intersection graph of* $\mathcal{S}$ has $\mathcal{S}$ as its vertex set and an edge whenever $S \cap S' \neq \emptyset$, for $S, S' \in \mathcal{S}$.

In the most general case, our objects are curves in the plane (which is equivalent to connected regions in the plane). We call the intersection graphs of curves in the plane *string graphs*. Ehrlich, Even, and Tarjan [EET76] proved in 1976 that not all graphs are string graphs and that not all string graphs are *segment graphs*, i.e., intersection graphs of straight-line segments in the plane. The proof idea that not all graphs are string graphs is as follows. Consider any non-planar graph G and subdivide each edge by a dummy vertex. Let G^+ denote the resulting graph.[3] Now assume for a contradiction that we have a *string representation*, i.e., a set of curves, of G^+ given. Modify this string representation by shrinking the curves representing the original vertices to points while extending the curves of the dummy vertices to preserve the intersections. Now the curves of the dummy vertices meet at the points of their adjacent original vertices, however, by construction, do not intersect each other. This is a planar drawing of the non-planar graph G – a contradiction. Further classical examples for the set $\mathcal{S}$ of geometric objects include disks, intervals on a line, and rectangles; see Figure 3.1 for two examples. For an overview of intersection graphs, see the books by McKee and McMorris [MM99] or Brandstädt, Le, and Spinrad [BLS99].

[3] We remark that G^+ is bipartite, which we use in Figure 3.2.



(a) String representation given as a set of curves in the plane and the corresponding string graph.

(b) Disk representation given as a set of circular disks in the plane and the corresponding disk graph.

Figure 3.1: Examples of intersection graphs of geometric objects in the plane.

Related Work. In the remainder of this chapter, we focus on the *recognition problem* for classes of intersection graphs of restricted geometric objects, i.e., determining whether a given graph is an intersection graph of a family of restricted sets of geometric objects. Consider the class of segment graphs mentioned before, which are the intersection graphs of line segments in the plane.[4] Chalopin and Gonçalves [CG09] proved that the segment graphs include planar graphs, and this proof has since then been simplified [GIP18]. However, the recognition problem for segment graphs is $\exists\mathbb{R}$-complete as shown by Kratochvíl and Matoušek [KM94, Mat14], which makes the problem harder than recognizing string graphs, which is "only" NP-complete [Kra91, SSS03].

On the other hand, one of the simplest natural subclasses of segment graphs is the class of the *permutation graphs*, the intersection graphs of line segments where there are two parallel lines such that each line segment has its two endpoints on these parallel lines.[5] We say that the segments are *grounded* on these two lines. The

[4] We follow the common convention that parallel segments do not intersect and each point in the plane belongs to at most two segments.

[5] I.e., we think of the sequence of endpoints on the "bottom" line as one permutation π of the vertices and the sequence on the "top" line as another permutation π', where uv is an edge if and only if the order of u and v differs in π and π'.

recognition problem for permutation graphs can be solved in linear time [KMMS06]. *Bipartite* permutation graphs have an even simpler intersection representation [SS94]: they are the intersection graphs of unit-length vertical and horizontal line segments which are again double-grounded (without loss of generality, both lines hosting the endpoints have slope -1). The simplicity of bipartite permutation graphs leads to a simpler linear-time recognition algorithm [SBS87] than that of permutation graphs.

Several recent articles [CJ17, CHO$^+$17, CFM$^+$18, CFHW18] compare and study the geometric intersection graph classes occurring between the simple classes, such as bipartite permutation graphs, and the general classes, such as string or segment graphs. For an overview, see also Figure 3.2.

When the segments are not grounded, but still are only horizontal and vertical, the class is referred to as *grid intersection graphs* and it also has a rich history [HNZ91, Kra94, CHO$^+$17, CFHW18]. In particular, note that the recognition problem is NP-complete for grid intersection graphs [Kra94]. On the positive side, if both the permutation of the vertical segments and the permutation of the horizontal segments are given, then the problem becomes a trivial check on the bipartite adjacency matrix [Kra94]. However, for the variant where only one such permutation, e.g., the order of the horizontal segments, is given, the complexity remains open. A few special cases of this problem have been solved efficiently [FMM13, CDK$^+$14, DHK$^+$19]. One such case [CDK$^+$14] is equivalent to the problem of *level planarity testing* which can be solved in linear time [JLM98].

Conversely, if the segments are grounded but not restricted to be horizontal and vertical, then we speak of *grounded segment graphs.* Similar to the general case of segment graphs, it was recently shown that the recognition problem for grounded segments is $\exists\mathbb{R}$-complete [CFM$^+$18].

Previous Work. In this chapter, we study recognition problems concerning so-called *stick* graphs, the intersection graphs of grounded vertical and horizontal line segments (i.e., grounded grid intersection graphs; see Figure 3.2). Classes closely related to stick graphs appear in several application contexts, e.g., in *nano PLA-design* [STTU11] and when detecting *loss of heterozygosity events in the human genome* [HATI11, CCF$^+$17].

Next, we formally define the problem of recognizing stick graphs.

Definition 3.1 (Stick)**.** Let G be a bipartite graph with vertex set $A\dot{\cup}B$, and let ℓ be a line with slope -1. Decide whether G has an intersection representation where the vertices in A are vertical line segments whose bottom endpoints lie on ℓ and the vertices in B are horizontal line segments whose left endpoints lie on ℓ.[6] Such a representation is a *stick representation* of G, the line ℓ is the *ground line*, the segments are called *sticks*, and the point where a stick meets ℓ is its *foot point.*

In 2018, stick graphs were introduced by Chaplick, Felsner, Hoffmann, and Wiechert [CFHW18]. The primary prior work on recognizing stick graphs is due to De Luca, Hossain, Kobourov, Lubiw, and Mondal [DHK$^+$19]. They introduced

[6] Note that De Luca et al. [DHK$^+$19] regarded A as the set of horizontal segments.

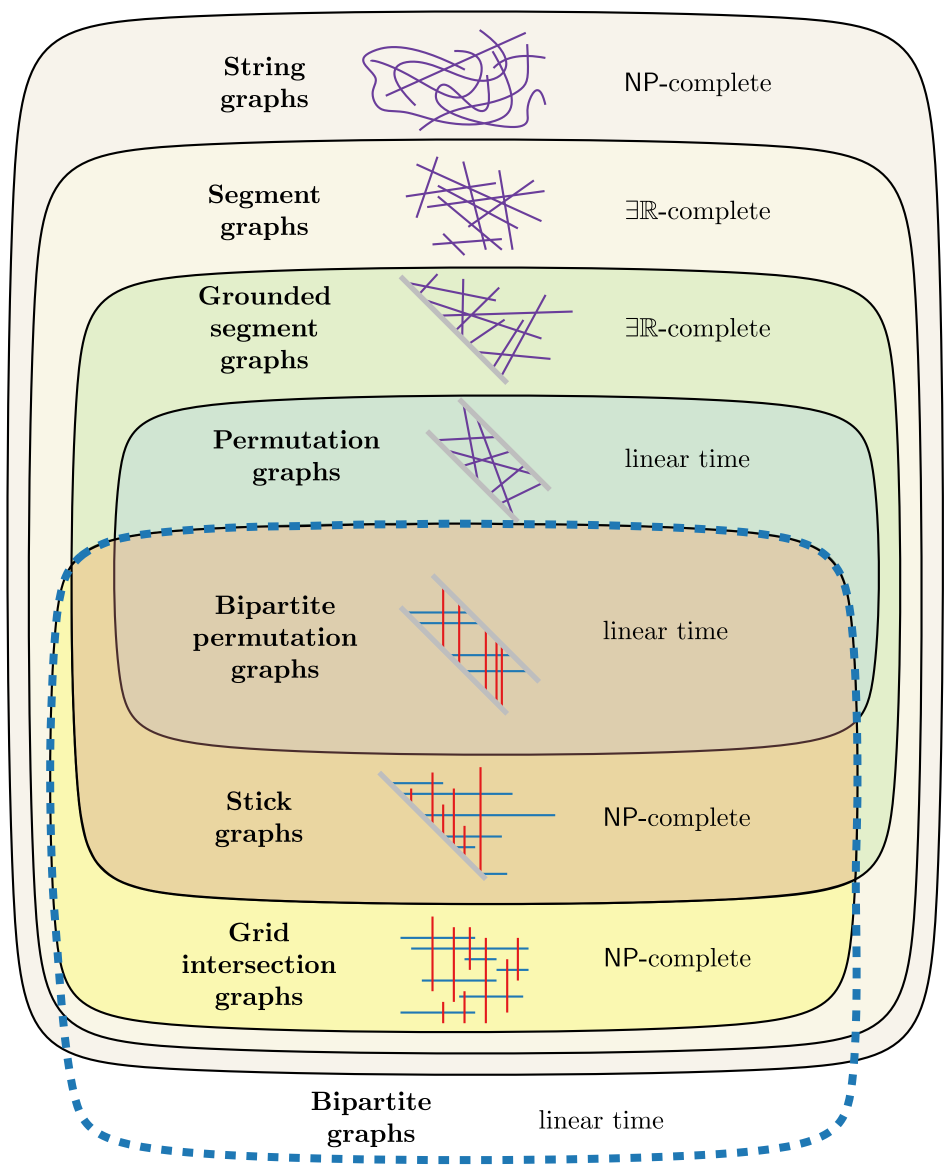


Figure 3.2: Euler diagram illustrating the relationship between various classes of intersection graphs and, for each graph class, the complexity of recognizing such a graph.

given order	variable length		fixed length: isolated vertices		fixed length: no isolated vertices	
$\emptyset$	NPc	[Rus23]	NPc	T3.5	NPc	T3.5
A	$\mathcal{O}(\lvert A\rvert + \lvert B\rvert + \lvert E\rvert)$	[Rus22]	NPc	T3.8	NPc	T3.8
A,B	$\mathcal{O}(\lvert A\rvert + \lvert B\rvert + \lvert E\rvert)$	T3.3	NPc	C3.9	$\mathcal{O}((\lvert A\rvert + \lvert B\rvert)^2)$	C3.13

Table 3.1: Time complexity for deciding whether a given bipartite graph $G = (A\dot{\cup}B, E)$ is a stick graph. We abbreviate NP-complete by NPc, Theorem by T, and Corollary by C. Blue entries mark the results presented in this chapter.

two constrained cases of the stick graph recognition problem called Stick$_A$ and Stick$_{AB}$, which we define next. Similarly to Kratochvíl's approach to grid intersection graphs [Kra94], De Luca et al. characterized stick graphs through their bipartite adjacency matrix and used this result as a basis to develop a polynomial-time algorithm to solve Stick$_{AB}$.

Definition 3.2 (Stick$_A$/Stick$_{AB}$)**.** In the problem Stick$_A$ (Stick$_{AB}$) we are given an instance of the Stick problem and additionally an order σ_A (orders σ_A, σ_B) of the vertices in A (in A and B). The task is to decide whether there is a stick representation that respects σ_A (σ_A and σ_B).

For Stick$_A$, De Luca et al. [DHK$^+$19] gave an $\mathcal{O}(|A| \cdot |B|)$-time algorithm in 2019. After that, we [CKL$^+$20] presented algorithms for Stick$_A$ and Stick$_{AB}$ with running times in $\mathcal{O}(|A|+|B|+|E|)$ and $\mathcal{O}(|A|\cdot|B|)$, respectively. Recently, Rusu has investigated the field of stick graphs [Rus20, Rus22, Rus23]. In 2022, she improved the running time of the recognition algorithm for Stick$_A$ to $\mathcal{O}(|A|+|B|+|E|)$ [Rus22]. Her approach is based on canonical orders of the vertices in A and B. More recently, she also showed that recognizing stick graphs is NP-complete [Rus23], which has been open so far.

Besides the general setting, we suggest using prescribed stick lengths as part of the input. Cabello and Jejčič [CJ17] mention that studying classes of intersection graphs with constraints on the sizes or lengths of the objects is an interesting direction for future work. Note that similar length restrictions have been considered for other geometric intersection graphs such as interval graphs [PS97, KKW15, KOS19].

Contribution. We first revisit the problems Stick$_A$ and Stick$_{AB}$ defined by De Luca et al. [DHK$^+$19]. We provide the first efficient algorithm for Stick$_A$[7] and a faster algorithm for Stick$_{AB}$; see Section 3.2. For our Stick$_A$ algorithm, we introduce a new tool, semi-ordered trees (see Section 3.2.2), as a way to capture all possible permutations of the horizontal sticks which occur in a solution to the given Stick$_A$ instance. This data structure may be of independent interest. Then, we investigate the direction suggested by Cabello and Jejčič [CJ17] where specific lengths are given for the segments of each vertex. In particular, this can be thought

[7] As mentioned above, there is now a faster algorithm published by Rusu [Rus22]. We still present this result since it covers nicely the structure of an instance of Stick$_A$.

of as generalizing from unit stick graphs (i.e., bipartite permutation graphs), where every segment has the same length. While bipartite permutation graphs can be recognized in linear time [SBS87], it turns out that all of the new problem variants (which we call $\textsc{Stick}^{\mathsf{fix}}$, $\textsc{Stick}^{\mathsf{fix}}_{\mathsf{A}}$, and $\textsc{Stick}^{\mathsf{fix}}_{\mathsf{AB}}$) are NP-complete; see Section 3.3. Finally, we give an efficient solution for $\textsc{Stick}^{\mathsf{fix}}_{\mathsf{AB}}$ (that is, $\textsc{Stick}_{\mathsf{AB}}$ with fixed stick lengths) for the special case that there are no isolated vertices (see Section 3.3.3). We conclude and state some open problems in Section 3.4. Our results are summarized in Table 3.1.

3.2 Sticks of Variable Lengths

In this section, we provide algorithms that solve the $\textsc{Stick}_{\mathsf{AB}}$ problem in $\mathcal{O}(|A|+|B|+|E|)$ time (see Section 3.2.1, Theorem 3.3) and the $\textsc{Stick}_{\mathsf{A}}$ problem in $\mathcal{O}(|A| \cdot |B|)$ time (see Section 3.2.3, Theorem 3.4). Between these subsections, in Section 3.2.2, we describe *semi-ordered trees*, an essential tool reminiscent of PQ-trees that we use for the latter algorithm. This tool will allow us to express the different ways one can order the horizontal segments for a given instance of $\textsc{Stick}_{\mathsf{A}}$.

3.2.1 Solving $\textsc{Stick}_{\mathsf{AB}}$ in $\mathcal{O}(|A| + |B| + |E|)$ Time

De Luca et al. [DHK+19] showed how to compute, for a given graph $G = (A \cup B, E)$ and orders σ_A and σ_B, a $\textsc{Stick}_{\mathsf{AB}}$ representation in $\mathcal{O}(|A| \cdot |B|)$ time (if such a representation exists). We improve upon their result in this section. Namely, we prove the following theorem.

Theorem 3.3. $\textsc{Stick}_{\mathsf{AB}}$ *can be solved in* $\mathcal{O}(|A| + |B| + |E|)$ *time.*

Proof. We apply a sweep-line approach (with a vertical sweep-line moving rightwards) where each vertical stick $a_i \in A$ corresponds to two events: the *enter event of* a_i (abbreviated by i) and the *exit event of* a_i (abbreviated by $i{\rightarrow}$).

Let $\sigma_A = (a_1, \dots, a_{|A|})$ and $\sigma_B = (b_1, \dots, b_{|B|})$. Let β_i denote the largest index such that b_{β_i} has a neighbor in $a_1, \dots, a_i$. Let $\hat{B}^i$ be the subsequence of $(b_1, \dots, b_{\beta_i})$ of those vertices that have a neighbor in $a_i, \dots, a_{|A|}$, and let $\hat{B}^{i\rightarrow}$ be the subsequence of $(b_1, \dots, b_{\beta_i})$ of those vertices that have a neighbor in $a_{i+1}, \dots, a_{|A|}$. At every event $p \in \{i, i{\rightarrow}\}$, we maintain the following invariants.

(i) We have a valid representation of the subgraph of G induced by the set of vertices $\{b_1, \dots, b_{\beta_i}, a_1, \dots, a_i\}$.

(ii) The x-coordinates of the foot points of $\{b_1, \dots, b_{\beta_i}, a_1, \dots, a_i\}$ are unique integers in the range from 1 to $\beta_i + i$.

(iii) For the vertices in $\{b_1, \dots, b_{\beta_i}, a_1, \dots, a_i\} \setminus \hat{B}^p$, both endpoints are set.

We initialize $\hat{B}^0 = \hat{B}^{0\rightarrow} = \emptyset$ and $\beta_0 = 0$. Here, our invariants trivially hold. Now suppose $i \geq 1$. In the following, we do not create a new ordered set $\hat{B}^p$ for each event p, but we update an ordered set $\hat{B}$, viewing $\hat{B}^p$ as the state of $\hat{B}$ during event p.

Consider the enter event of a_i. We set the x-coordinate of a_i to $\beta_i + i$. We place the foot points of vertices $b_{\beta_{i-1}+1}, \dots, b_{\beta_i}$ (if they exist) between a_{i-1} and a_i in this order and create $\hat{B}^i$ by appending them to $\hat{B}^{(i-1)\rightarrow}$ in this order. All neighbors of a_i have to start before a_i, and they have to be a suffix of $\hat{B}^i$. If this is not the case, then we simply reject as this is a negative instance of the problem. This is easily checked in $\mathcal{O}(\deg(a_i))$ time. The upper endpoint of a_i is placed half a unit above the foot point of its first neighbor in this suffix. As such, the invariants (i)–(iii) are maintained.

Consider the exit event of a_i. For each neighbor b_j of a_i, we check whether a_i is the last neighbor of b_j in σ_A. In this case, we remove b_j from $\hat{B}$ and we finish b_j by setting the x-coordinate of its right endpoint to $\beta_i + i + 1/2$. Now $\hat{B}^{i\rightarrow}$ consists of all vertices in $\hat{B}^i$ except those that we just finished. Again, this maintains our invariants (i)–(iii). Note that processing the exit event always succeeds, i.e., negative instances are detected purely in the enter events.

Hence, if we reach and complete the exit event of $a_{|A|}$, we obtain a STICK$_{\mathsf{AB}}$ representation of G. Otherwise, G has no such representation.

Finally, let us consider the runtime. We maintain the ordered set $\hat{B}$ as a doubly connected list. We iterate over all vertices in $|A|$ and, for every vertex of $|B|$, we add and remove it to/from $\hat{B}$ exactly once. Moreover, in the enter events, we check the last entries $\hat{B}$ – once for each edge. This results in an $\mathcal{O}(|A| + |B| + |E|)$ runtime.

Note that even though we have not explicitly discussed isolated vertices, these can be easily realized by sticks of length 1/2. $\square$

3.2.2 Data Structure: Semi-Ordered Trees

In the STICK$_{\mathsf{A}}$ problem, the goal is to find a permutation of the horizontal sticks B that is consistent with the fixed permutation of the vertical sticks A. To this end, we make use of a data structure that allows us to capture many permutations subject to consecutivity constraints. This might remind the reader of other similar but distinct data structures such as PQ-trees [BL76].

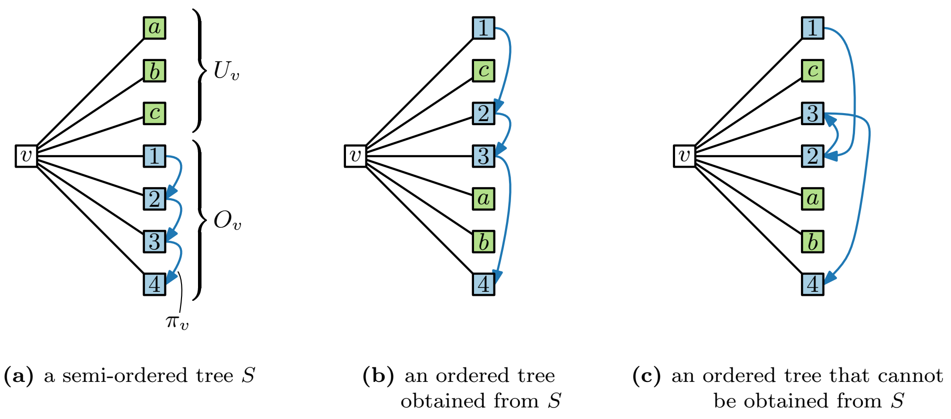
An *ordered tree* is a rooted tree where the order of the children of each internal node is specified. The permutation *expressed* by an ordered tree T is the permutation of its leaves in the pre-order traversal of T. Generalizing this, we define a *semi-ordered tree* where, for each node, there is a fixed permutation for a subset of the children and the remaining children are free. Namely, for each node v, we have

(i) a set U_v of *unordered* children,

(ii) a set O_v of *ordered* children, and

(iii) a fixed permutation π_v of O_v; see Figure 3.3.

Hence, every node (except the root) is ordered or unordered depending on its parent. We *obtain* an ordered tree from a semi-ordered tree by fixing, for each node v, a permutation π'_v of $O_v \cup U_v$ that contains π_v as a subsequence. In this way, a permutation π is *expressed* by a semi-ordered tree S if there exists an ordered tree T that expresses π and can be obtained from S.



a
b
c
U_v
v
1
2
3
4
O_v
π_v
(a) a semi-ordered tree S
1
c
2
3
a
b
4
v
(b) an ordered tree obtained from S
1
c
3
2
a
b
4
v
(c) an ordered tree that cannot be obtained from S

Figure 3.3: Definition of a semi-ordered tree.

3.2.3 Solving Stick_{A} in $\mathcal{O}(|A| \cdot |B|)$ Time

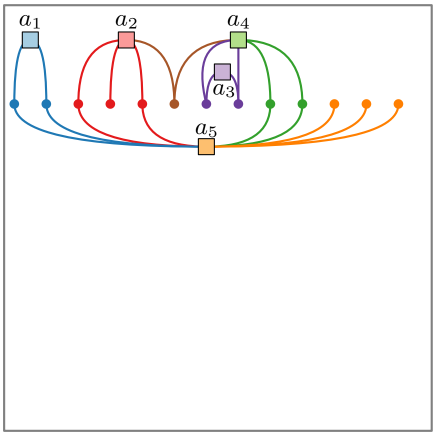
Let $G = (A\dot{\cup}B, E)$ and $\sigma_A = (a_1, \ldots, a_{|A|})$ be the input. We assume that G is connected and discuss otherwise at the end of this section.

As in the algorithm for Stick_{AB}, we apply a sweep-line approach (with a vertical sweep-line moving rightwards) where again each vertical stick $a_i \in A$ corresponds to two events: the *enter event of* a_i (abbreviated by i) and the *exit event of* a_i (abbreviated by $i\!\rightarrow$).

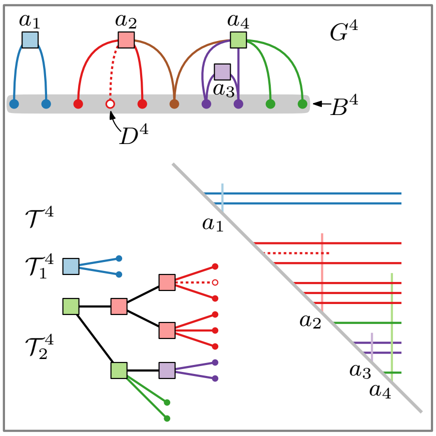
Overview. Informally, for each event $p \in \{i, i\!\rightarrow\}$, we maintain all representations of the subgraph seen so far subject to certain horizontal sticks continuing further (those that intersect the sweep-line and some vertical stick before it). We denote by G^p the induced subgraph of G containing $a_1, \ldots, a_i$ and their neighbors in B. We distinguish the neighbors in B as those that are *dead* (i.e., have all neighbors before the sweep-line) and those that are *active* (i.e., have neighbors before and after the sweep-line). Namely,

- B^p consists of all sticks of B in G^p;
- D^i consists of all (dead) sticks of B^i with no neighbor in $a_i, \ldots, a_{|A|}$; and
- $D^{i\rightarrow}$ consists of all (dead) sticks of $B^{i\rightarrow}$ with no neighbor in $a_{i+1}, \ldots, a_{|A|}$.

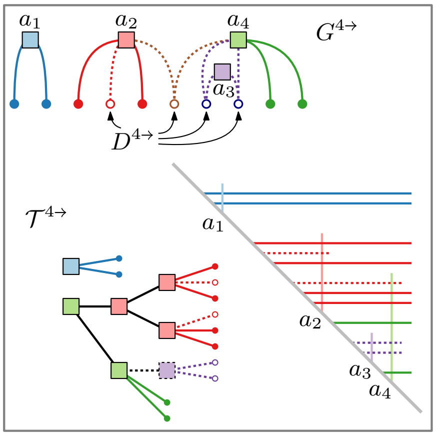
To this end, we maintain an ordered forest $\mathcal{T}^p$ of special semi-ordered trees that encodes all realizable permutations (defined below) of the set of horizontal sticks B^p as the permutations expressed by $\mathcal{T}^p$; see Figure 3.4. A permutation π of B^p is *realizable* if there is a stick representation of the graph H^p obtained from G^p by adding a vertical stick to the right of a_i neighboring all active horizontal sticks in B^p where B^p is drawn top-to-bottom in order π.



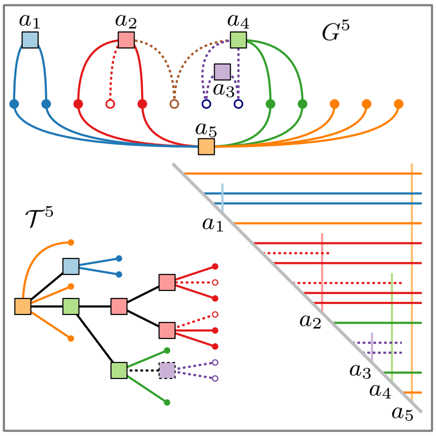
(a) the graph G



(b) enter event of a_4



(c) exit event of a_4



(d) enter event of a_5

Figure 3.4: An example for the data structures and a drawing corresponding to a realizable permutation of B^p for a graph G with $\pi_a = (a_1, \ldots, a_k)$. Vertices in A are drawn as squares; vertices in B are drawn as disks. The vertices and edges are colored by the entering event in which they appear. Dead vertices are drawn as empty disks; their edges are dotted. In (d), the leaves are permuted in the order in which they are drawn.

In the enter event of a_i, B^i comprises $B^{i-1\rightarrow}$ and all vertices of B that neighbor a_i and are not in the data structure yet (we call these *entering vertices*). We constrain the data structure so that all the neighbors of a_i must occur after (below) the non-neighbors of a_i. The set D^p of dead vertices remains unchanged with respect to the last exit event, that is, $D^i = D^{(i-1)\rightarrow}$.

In the exit event of a_i, $D^{i\rightarrow}$ comprises D^i and all sticks of B^i that do not have any neighbor a_j with $j > i$, i.e., those having a_i as their last neighbor (we call these *exiting vertices*). No new horizontal sticks appear in an exit event, hence $B^{i\rightarrow} = B^i$.

Data Structure. See Figure 3.4 for an example. Consider any event p. Observe that G^p may consist of several connected components $G^p_1, \ldots, G^p_{k_p}$. The components of G^p are naturally ordered from left to right by σ_A. Let B^p_j denote the vertices of B^p in G^p_j. In this case, in every realizable permutation of B^p, the vertices of B^p_j must come before the vertices of B^p_{j+1}. Furthermore, the vertices that are introduced any time later can only be placed at the beginning, end, or between the components. Hence, to compactly encode the realizable permutations, it suffices to do so for each component G^p_j individually via a semi-ordered tree T^p_j. Namely, our data structure is $\mathcal{T}^p = (T^p_1, \ldots, T^p_{k_p})$. Each data structure T^p_j is a special semi-ordered tree in which the leaves correspond to the vertices of B^p_j, all leaves are unordered, and all internal vertices are ordered.

Correctness and Event Processing. We argue by induction that this data structure is sufficient to express the realizable permutations of B^p. We maintain the following invariants for each event p during the execution of the algorithm.

(I1) The set of permutations expressed by $\mathcal{T}^p$ contains all permutations of B^p which occur in a stick representation of G.

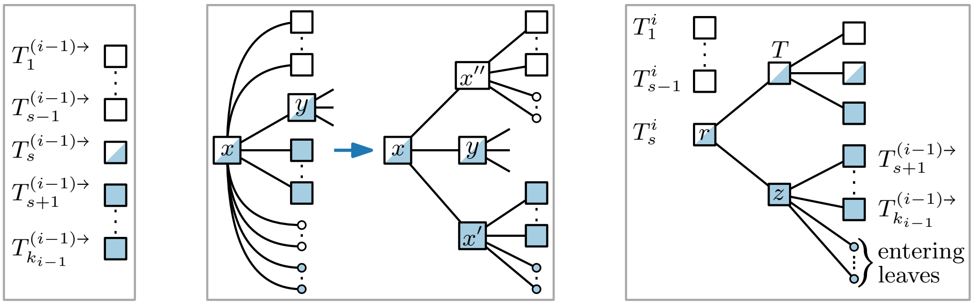
(I2) The set of permutations expressed by $\mathcal{T}^p$ contains only permutations of B^p which occur in a stick representation of G^p.

Since $G^{|A|\rightarrow} = G$ and $B^{|A|\rightarrow} = B$, after the final step these invariants ensure that our data structure expresses exactly those permutations of B which occur in a stick representation of G.

Recall that our data structure consists of an ordered set of semi-ordered trees. Note that these invariants also apply to each semi-ordered tree individually, that is, to its corresponding connected component.

In the base case, consider the enter event of a_1. Our data structure consists of a single component G^1_1 and clearly, a single node with a leaf-child for every neighbor of a_1 captures all possible permutations.

In the exit event of a_i, we do not change the shape of $\mathcal{T}^i$ because only the set of dead vertices is modified. Hence, $\mathcal{T}^{i\rightarrow} = \mathcal{T}^i$ and our two invariants remain correct. However, as an additional algorithmic step in $\mathcal{T}^{i\rightarrow}$, we mark the exiting vertices as dead and add them to $D^{i\rightarrow}$. We further mark any internal node in $\mathcal{T}^{i\rightarrow}$ that contains only dead leaves in its subtree as dead as well. Obviously, this procedure maintains all the invariants.



$T_1^{(i-1)\rightarrow}$
$T_{s-1}^{(i-1)\rightarrow}$
$T_s^{(i-1)\rightarrow}$
$T_{s+1}^{(i-1)\rightarrow}$
$T_{k_{i-1}}^{(i-1)\rightarrow}$
x
y
x''
x'
T_1^i
T_{s-1}^i
T_s^i
T
r
z
$T_{s+1}^{(i-1)\rightarrow}$
$T_{k_{i-1}}^{(i-1)\rightarrow}$
entering leaves

(a) $\mathcal{T}^{(i-1)\rightarrow}$ **(b)** transformation of T **(c)** $\mathcal{T}^i$

Figure 3.5: Construction of $\mathcal{T}^i$. Leaves are drawn as small disks. The leaves at the new node z are the entering vertices. Only active vertices are shown.

Now consider the enter event of a_i and assume that we have the data structure $\mathcal{T}^{(i-1)\rightarrow} = (T_1^{(i-1)\rightarrow}, \dots, T_{k_{i-1}}^{(i-1)\rightarrow})$. The essential observation is that the neighbors of a_i must form a suffix of the active vertices in $B^{(i-1)\rightarrow}$ in every realizable permutation after the enter event, which we enforce in the following. Namely, either

- all active vertices in $B^{(i-1)\rightarrow}$ are adjacent to a_i,
- none of them are adjacent to a_i, or
- there is an $s \in \{1, \dots, k_{i-1}\}$ such that
  - (i) $B_s^{(i-1)\rightarrow}$ contains at least one neighbor of a_i;
  - (ii) all active vertices in $B_{s+1}^{(i-1)\rightarrow}, \dots, B_{k_{i-1}}^{(i-1)\rightarrow}$ are neighbors of a_i; and
  - (iii) no active vertices in $B_1^{(i-1)\rightarrow}, \dots, B_{s-1}^{(i-1)\rightarrow}$ are adjacent to a_i; see Figure 3.5a.

Otherwise, there is no realizable permutation for this event and consequently for G. The first two cases can be seen as degenerate cases (with $s = 0$ or $s = k_{i-1} + 1$) of the general case, which we analyze next.

We first show how to process $T_s^{(i-1)\rightarrow}$; see Figure 3.5b. Afterwards, we describe how to construct the data structure $\mathcal{T}^i$.

We create a tree T that expresses precisely the subset of the permutations expressed by $T_s^{(i-1)\rightarrow}$ where all leaves that are neighbors of a_i occur as a suffix. We initialize $T = T_s^{(i-1)\rightarrow}$. If all active vertices in $B_s^{(i-1)\rightarrow}$ are neighbors of a_i, then we are already done.

Otherwise, we say that a node of T is *marked* if all active leaves in its subtree are neighbors of a_i; it is *unmarked* if no active leaf in its subtree is a neighbor of a_i; and it is *half-marked* otherwise. Note that the root of T is half-marked. (We can ignore the dead nodes and leaves.)

Since the neighbors of a_i must form a suffix of the active leaves, the marked non-leaf children of a half-marked node form a suffix of the active children, the unmarked non-leaf children form a prefix of the active children, and there is at most one half-marked child. Hence, the half-marked nodes form a path in T that starts in the root; otherwise, there are no realizable permutations for this event and subsequently for G.

We traverse the path starting from the deepest descendant and ending at the root, i.e., bottom-to-top. Let x be a half-marked node, and let y be its half-marked child (if it exists); see Figure 3.5b. We have to enforce that in any ordered tree obtained from T, the unmarked children of x occur before y and the marked children of x occur after y. We create a new marked vertex x'. This vertex receives the following children from x: the marked leaf-children and the suffix of the non-leaf children starting after y. Symmetrically, we create a new unmarked vertex x'', which receives the following children from x: the unmarked leaf-children and the prefix of the non-leaf children ending before y. Then we make x' and x'' children of x such that x'' is before y and y is before x'. If this results in any internal node having no leaf-children and only one child, we merge this node with its parent. (Note that this can only happen to x' or x''.) This ensures that for every permutation expressed by T, the subsequence of active vertices has the neighbors of a_i as a suffix.

Note that every non-leaf of $T_s^{(i-1)\rightarrow}$ is also a non-leaf in T with the same set of leaves in its subtree. In the pre-order traversal of any ordered tree obtained from T, the non-leaves of $T_s^{(i-1)\rightarrow}$ are visited in the same order as in the pre-order traversal of any ordered tree obtained from $T_s^{(i-1)\rightarrow}$. This implies that each permutation expressed by T is also expressed by $T_s^{(i-1)\rightarrow}$. Consequently, invariant (I2) holds locally for T.

The marked leaf-children of any half-marked node x of $T_s^{(i-1)\rightarrow}$ can be placed anywhere before, between, or after its marked children, but not before y (since y has both marked and unmarked children). Symmetrically, the unmarked leaf-children of any half-marked node x of $T_s^{(i-1)\rightarrow}$ can be placed anywhere before, between, or after its unmarked children, but not after y. Hence, each permutation expressed by $T_s^{(i-1)\rightarrow}$ that has the neighbors of a_i as a suffix of the subsequence of its active vertices is also expressed by T. Therefore, invariant (I1) holds locally for T.

Thus, the permutations expressed by T are exactly those expressed by $T_s^{(i-1)\rightarrow}$ that have the neighbors of a_i as a suffix of their active subsequence.

Now, we create the data structure $\mathcal{T}^i$; see Figure 3.5c. We set $T_1^i = T_1^{(i-1)\rightarrow}, \dots, T_{s-1}^i = T_{s-1}^{(i-1)\rightarrow}$. Clearly, both invariants hold locally for $T_1^i, \dots, T_{s-1}^i$. Next, we create a new semi-ordered tree T_s^i as follows. The tree T_s^i gets a new root r, to which we attach T and a new vertex z, in this order. Then we hang $T_{s+1}^{(i-1)\rightarrow}, \dots, T_{k_{i-1}}^{(i-1)\rightarrow}$ from z in this order. We further make the entering vertices leaf-children of z. Note that this allows them to mix freely before, after, or between the components $G_{s+1}^{(i-1)\rightarrow}, \dots, G_{k_{i-1}}^{(i-1)\rightarrow}$. In the special case that all active leaves of T are neighboring a_i, we append T to z instead of r (we then merge r and z) to allow the entering vertices also to appear before T. Finally, we set $\mathcal{T}^i = (T_1^i, \dots, T_s^i)$.

In this way, the order of the components $G_1^{(i-1)\rightarrow}, \dots, G_{k_{i-1}}^{(i-1)\rightarrow}$ of $G^{(i-1)\rightarrow}$ is maintained in the data structures for G^i. In T_s^i, both invariants clearly hold for the non-leaf children of z and, as argued above, also for T. Furthermore, we ensure that the entering vertices can be placed exactly before, after, or between the components of $G^{(i-1)\rightarrow}$ that are completely adjacent to a_i. Hence, both invariants hold for $\mathcal{T}^i$.

The decision problem of $\textsc{Stick}_\mathsf{A}$ can easily be solved by this algorithm. Namely, by our invariants, any permutation σ_B expressed by $\mathcal{T}^{|A|\rightarrow}$ occurs as a permutation of the horizontal sticks in a $\textsc{Stick}_\mathsf{A}$ representation of G. Thus, executing our algorithm for $\textsc{Stick}_\mathsf{AB}$ on σ_A and σ_B gives us a stick representation of G. This concludes the proof of correctness for the connected case.

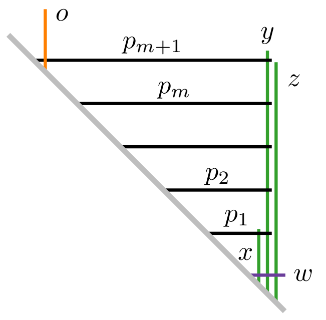
Disconnected Graphs. To handle disconnected graphs, we first identify the connected components $H_1, \dots, H_t$ of G. We label each element of A by the index of the component to which it belongs. Now, observe that if σ_A contains a pattern of indices a and a' that alternate as in $aa'aa'$, then the given $\textsc{Stick}_\mathsf{A}$ instance does not admit a solution. Otherwise, we can treat each component separately by our algorithm, and then nest the resulting representations. Namely, for each connected component H_r, we run our $\textsc{Stick}_\mathsf{A}$ algorithm (on σ_A restricted to H_r) to obtain a realizable permutation σ_{B_r} of the horizontal sticks of H_r. Now, since our connected components avoid the pattern $aa'aa'$, there is a natural hierarchy of these components which we can use to obtain a total order σ_B on the horizontal sticks of G from the permutations $\sigma_{B_1}, \dots, \sigma_{B_t}$. Finally, we can use this σ_B, the given σ_A, and G as an input to our $\textsc{Stick}_\mathsf{AB}$ algorithm to obtain a representation.

Running Time. The size of each data structure $\mathcal{T}^p$ is in $\mathcal{O}(|B^p|)$ since there are no degree-2 vertices in the trees and each leaf corresponds to a vertex in B^p. In each event, the transformations can clearly be done in time proportional to the size of the data structures. Since $|B^p| \le |B|$ for each p and there are $2|A|$ events, we get the following running time.

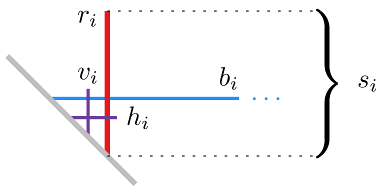
Theorem 3.4. *$\textsc{Stick}_\mathsf{A}$ can be solved in $\mathcal{O}(|A| \cdot |B|)$ time.*

3.3 Sticks of Fixed Lengths

In this section, we consider the case that, for each vertex of the input graph, its stick length is fixed – more precisely, it is part of the input. We denote the variants of this problem by $\textsc{Stick}^\mathsf{fix}$ if the order of the sticks is not restricted, by $\textsc{Stick}^\mathsf{fix}_\mathsf{A}$ if σ_A is given, and by $\textsc{Stick}^\mathsf{fix}_\mathsf{AB}$ if σ_A and σ_B are given. Unlike in the case with variable stick lengths, all three new variants are NP-hard; see Sections 3.3.1 and 3.3.2. Surprisingly, $\textsc{Stick}^\mathsf{fix}_\mathsf{AB}$ can be solved efficiently by a simple linear program if the input graph contains no isolated vertices (i.e., vertices of degree 0); see Section 3.3.3. With our linear program, we can check the feasibility of any instance of $\textsc{Stick}^\mathsf{fix}$ if we are given a total order of the sticks on the ground line. Together with our NP-hardness results, this implies NP-completeness of $\textsc{Stick}^\mathsf{fix}$, $\textsc{Stick}^\mathsf{fix}_\mathsf{A}$, and $\textsc{Stick}^\mathsf{fix}_\mathsf{AB}$.



(a) frame providing the pockets



(b) number gadget for the number s_i

Figure 3.6: Gadgets of our reduction from 3-PARTITION to STICK$^{\mathsf{fix}}$.

3.3.1 STICK$^{\mathsf{fix}}$

We show that STICK$^{\mathsf{fix}}$ is NP-hard by reduction from 3-PARTITION, which is strongly NP-hard [GJ79]. In 3-PARTITION, one is given a multiset S of $3m$ integers $s_1, \dots, s_{3m}$ such that, for each $i \in \{1, \dots, 3m\}$, $C/4 < s_i < C/2$, where $C = (\sum_{i=1}^{3m} s_i)/m$, and the task is to decide whether S can be split into m sets of three integers, each summing up to C exactly.

Theorem 3.5. STICK$^{\mathsf{fix}}$ *is NP-complete.*

Proof. As mentioned at the beginning of this section, NP-membership follows from our linear program (see Theorem 3.12 in Section 3.3.3) to test the feasibility of any instance of STICK$^{\mathsf{fix}}$ when given a total order of the sticks on the ground line.

To show NP-hardness, we describe a polynomial-time reduction from 3-PARTITION to STICK$^{\mathsf{fix}}$. Given a 3-PARTITION instance $I = (S, C, m)$, we construct a fixed cage-like frame structure and introduce a number gadget for each number of S.

The Frame. A sketch of the frame is given in Figure 3.6a. The purpose of the frame is to provide pockets, which host our number gadgets (defined below). To construct the frame, we add two long vertical (green) sticks y and z of length $mC + 1 + 2\varepsilon$, where $\varepsilon \ll 1$, and a shorter vertical (green) stick x of length 1 that are all kept together by a short horizontal (violet) stick w of length ε. We use $m + 1$ horizontal (black) sticks $p_1, p_2, \dots, p_{m+1}$ to separate the pockets. Each of them intersects y but not z and has a specific length such that the vertical distance between each two neighboring sticks p_i, p_{i+1}, where $i \in \{1, \dots, m\}$, is $C \pm \varepsilon$.

Additionally, p_1 intersects x and p_{m+1} intersects a vertical (orange) stick o of length $2C$. We use x and o to prevent the number gadgets from being placed below the bottommost and above the topmost pocket, respectively. It does not matter on which side of the stick y the stick x ends up since each b_i of a number gadget intersects y but neither x nor z.

The Number Gadgets. For each number s_i in S, we construct a number gadget; see Figure 3.6b. We introduce a vertical (red) stick r_i of length s_i. Intersecting stick r_i, we add a horizontal (blue) stick b_i of length $mC + 2$. The stick b_i intersects the sticks y and z, but neither x nor o. Due to these adjacencies, every number gadget can only be placed in one of the m pockets defined by $p_1, \dots, p_{m+1}$. It cannot span multiple pockets. We require that stick r_i and stick b_i intersect each other close to their foot points, so we introduce two short (violet) sticks h_i and v_i – one horizontal, the other vertical – of lengths ε; they intersect each other, h_i intersects r_i, and v_i intersects b_i.

Correctness. Given a yes-instance $I = (S, C, m)$ of 3-Partition together with a valid 3-partition P of S, the graph obtained by our reduction is realizable. Construct the frame as described before and place the number gadgets into the pockets according to P. Consider some pocket and say the three number gadgets placed there correspond to numbers s_i, s_j, s_k for some $i, j, k \in \{1, \dots, 3m\}$. The lengths of the sticks r_i, r_j, and r_k sum up to $C \pm 3\varepsilon$. Thus, the three number gadgets of s_i, s_j, and s_k can be placed together into one pocket since each pocket has height $C \pm \varepsilon$. After distributing all number gadgets, we have a stick representation.

On the other hand, given a stick representation of a graph G obtained from our reduction, we can obtain a valid solution of the corresponding 3-Partition instance $I = (S, C, m)$ as follows. Clearly, the shape of the frame is fixed, creating m pockets. Since the sticks $b_1, \dots, b_{3m}$ are incident to y and z but neither to x nor to o, they can end up inside any of the pockets. In the y-dimension, each two number gadgets of numbers s_i and s_j overlap at most on a section of length ε; otherwise r_i and b_j or r_j and b_i would intersect. Each pocket hosts precisely three number gadgets: we have $3m$ number gadgets, m pockets, and no pocket can contain four (or more) number gadgets; otherwise, there would be a number gadget of height at most $(C+\varepsilon)/4 + 2\varepsilon$, contradicting the fact that, for each $i \in \{1, \dots, 3m\}$, s_i is an integer with $s_i > C/4$ (so, $s_i \geq C/4 + 1 > (C+\varepsilon)/4 + 2\varepsilon$). Moreover, in each pocket, the total height of the three number gadgets would be too large if the three corresponding numbers of S would sum up to $C + 1$ or more. Thus, the assignment of number gadgets to pockets defines a valid 3-partition of S. □

The sticks of lengths $s_1, \dots, s_{3m}$ can be simulated by paths of sticks with length ε each. We refer to them as *ε-paths*; see Figures 3.7 and 3.8. An *ℓ-vertex ε-path* is a sequence of (violet) sticks $e_1, \dots, e_\ell$ of length ε that is alternating between horizontal and vertical sticks and, for each $i \in \{1, \dots, \ell - 1\}$, the sticks e_i and e_{i+1} intersect. Employing ε-paths in our reduction, it suffices to use only three distinct stick lengths. Like a spring, an ε-path can be stretched (Figure 3.7a) and compressed (Figure 3.7c) up to a specific length. We will exploit the following properties regarding the minimum and the maximum size of an ε-path.

Lemma 3.6. *There is a stick representation of a $2n$-vertex ε-path with height and width $n\varepsilon$ and another stick representation with height and width $\frac{n+2}{3}\varepsilon + \delta$ for any $\delta > 0$ and $n \geq 3$. Any stick representation of a $2n$-vertex ε-path has height and width in the range $\left(\frac{n-1}{3}, n\right] \varepsilon$.*

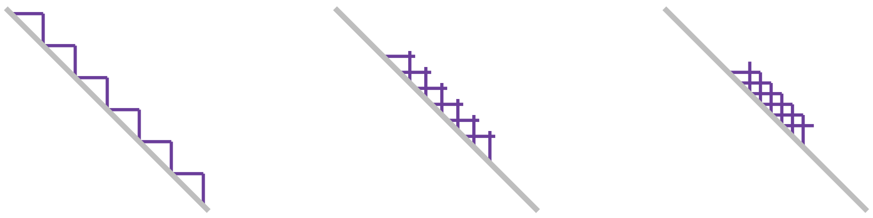


Figure 3.7: Three stick representations of an ε-path with twelve sticks.

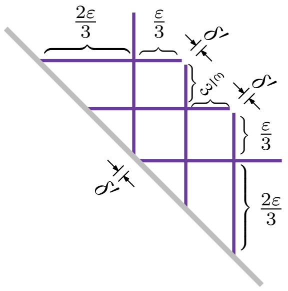


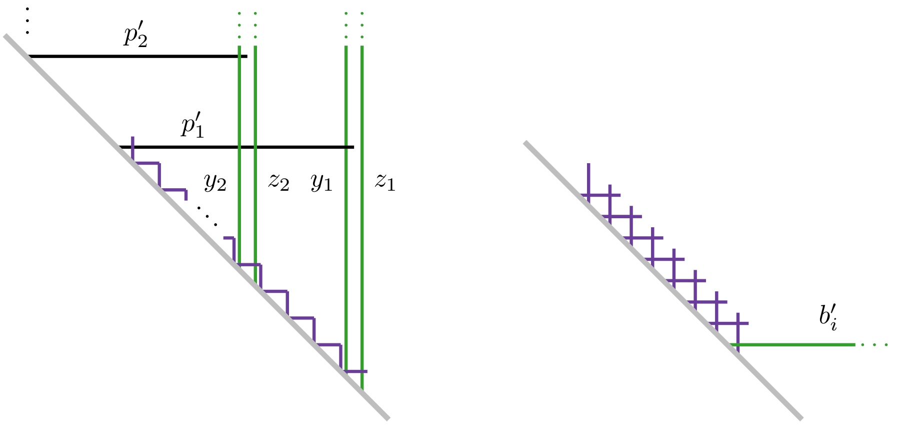
Figure 3.8: Construction of a compressed stick representation of an ε-path

Proof. We can arrange our sticks such that the foot points or the endpoints of two adjacent sticks touch each other (see Figure 3.7a). This construction has height and width $n\varepsilon$ and, clearly, this is the maximum width and height for a $2n$-vertex ε-path.

For the compressed ε-paths, we first describe a construction that has the specified width and height and, second, we show the lower bound.

The following construction is depicted in Figure 3.8 for $n = 3$. We define $\delta' = \delta/(n-2)$. Set the foot point of the first vertical stick – w.l.o.g. let this be e_1 – to $y = 0$ and the foot point of e_3, which is also vertical, to $y = \varepsilon/3$. For each $i \in \{2, \dots, n-1\}$, set the foot point of e_{2i-2}, which is horizontal, to $y = i\varepsilon/3 + (i-2)\delta'$ and the foot point of e_{2i+1}, which is vertical, to $y = i\varepsilon/3 + (i-1)\delta'$. Set the foot point of e_{n-2} to $y = n\varepsilon/3 + \delta$, and the foot point of e_n to $y = (n+1)\varepsilon/3 + \delta$. Observe that this construction has width and height $\frac{n+2}{3}\varepsilon + \delta$ and is a valid stick representation of a $2n$-vertex ε-path.

Now let us show the lower bound. For any $i \in \{4, \dots, 2n-4\}$, consider the line L through the stick e_i of the $2n$-vertex ε-path. On the one side of L, there is e_{i-3}, and on the other, there is e_{i+3}. For example, e_2 is to the right of e_5 and e_8 is to the left of e_5. Since all sticks have length ε and non-adjacent sticks are not allowed to touch each other, e_1, e_4, e_7, …, e_{2n} form a zigzag chain of width and height strictly greater than $\lfloor \frac{2n+2}{6} \rfloor \varepsilon \geq \frac{n-1}{3}\varepsilon$. $\square$



(a) frame providing the pockets **(b)** number gadget for the number s_i

Figure 3.9: Gadgets of our reduction from 3-Partition to Stick$^{\text{fix}}$ with three stick lengths.

Corollary 3.7. Stick$^{\text{fix}}$ *with only three different stick lengths is NP-complete.*

Proof. We modify the reduction from 3-Partition to Stick$^{\text{fix}}$ described in the proof of Theorem 3.5 such that we use only three distinct stick lengths. We use the three lengths ε, mC, and $3mC$ (or longer, e.g. ∞). In Figure 3.9, sticks of these lengths are violet, black, and green, respectively.

First, we describe the modifications of the frame structure, which are also depicted in Figure 3.9a. Instead of the vertical (green) sticks x, y, and z used to fix all pockets, we have two vertical sticks y_j and z_j of length $3mC$ for $j \in \{1, \dots, m+1\}$. Instead of the sticks $p_1, \dots, p_{m+1}$ of different lengths, we use horizontal (black) sticks $p'_1, \dots, p'_{m+1}$ each with length mC to separate the pockets. The stick p'_j intersects y_k, z_k for all $k \in \{j+1, \dots, m+1\}$ and y_j but not z_j. All pairs (y_j, z_j) are kept together by a stick of length ε. For each two neighboring pairs (y_j, z_j) and (y_{j+1}, z_{j+1}), these sticks of length ε are connected by an $(2C/\varepsilon)$-vertex ε-path. According to Lemma 3.6, this effects a maximum distance of $\leq (C/\varepsilon) \cdot \varepsilon = C$ between each two pairs of (y_j, z_j) and (y_{j+1}, z_{j+1}). Accordingly, the pockets separated by the sticks $p'_1, \dots, p'_{m+1}$ have height at most $C + 2\varepsilon$ and there is a realization such that every pockets has height C, which is similar as in the proof of Theorem 3.5. We keep the vertical (orange) stick o as in Figure 3.6a to prevent number gadgets from being placed above the topmost pocket, but now o has length $3Cm$.

Second, we describe the modifications of the number gadgets for each number s_i for $i \in \{1, \dots, 3m\}$, which are also depicted in Figure 3.9b. We keep a long stick b'_i similar to the stick b_i – now with length $3Cm$ and adjacent to each y_j and z_j for $j \in \{1, \dots, m+1\}$. We replace the stick r_i (together with the sticks h_i and v_i) by a $(6s_i/\varepsilon - 4)$-vertex ε-path. We make the first stick of the ε-path intersect stick b'_i. By Lemma 3.6, this ε-path has a stick representation with height $s_i + \delta$ for any $\delta > 0$, but there is no stick representation with height $s_i - \varepsilon$ or smaller. Clearly, to intersect all y- and z-sticks, the number gadgets can only be placed into the pockets of the frame. Moreover, none of their sticks can intersect a p'_j for $j \in \{1, \dots, m+1\}$ and they cannot intersect each other.

Hence, we can represent a yes-instance of 3-Partition as such a stick graph if and only if the ε-paths of the number gadgets are (almost) compressed as much as possible (to make the number gadgets small enough) and the ε-paths between the y- and z-sticks are (almost) stretched as much as possible (to have sufficiently high pockets). Using this, the proof is the same as in Theorem 3.5. □

3.3.2 $\textsc{Stick}^{\mathsf{fix}}_{\mathsf{A}}$ and $\textsc{Stick}^{\mathsf{fix}}_{\mathsf{AB}}$

We have seen that, deciding whether there is a stick representation when the stick lengths are given is as hard as solving the original problem Stick. Therefore, we next consider restricted versions of $\textsc{Stick}^{\mathsf{fix}}$, namely $\textsc{Stick}^{\mathsf{fix}}_{\mathsf{A}}$ and $\textsc{Stick}^{\mathsf{fix}}_{\mathsf{AB}}$, where σ_A and (σ_A, σ_B) are given, respectively. By analogy with Stick to $\textsc{Stick}_{\mathsf{A}}$ and $\textsc{Stick}_{\mathsf{AB}}$, one could expect that, while $\textsc{Stick}^{\mathsf{fix}}$ is NP-complete, $\textsc{Stick}^{\mathsf{fix}}_{\mathsf{A}}$ and $\textsc{Stick}^{\mathsf{fix}}_{\mathsf{AB}}$ are polynomial-time solvable. However, this is not true – both variants are NP-complete and, thus, harder than the corresponding problems $\textsc{Stick}_{\mathsf{A}}$ and $\textsc{Stick}_{\mathsf{AB}}$ where no stick lengths are given as part of the input.

We remark, that in some sense $\textsc{Stick}^{\mathsf{fix}}_{\mathsf{AB}}$ is actually a little simpler because only “unnatural” instances with isolated vertices are NP-complete, while all other instances are polynomial-time solvable, which we show in Section 3.3.3. On the other hand, we also show in this section that for $\textsc{Stick}^{\mathsf{fix}}_{\mathsf{A}}$, it does not help to forbid isolated vertices.

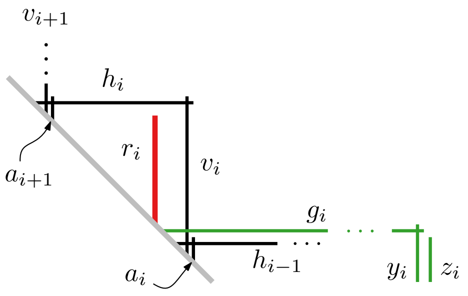
The NP-hardness reduction, which we present in this section, is almost the same for $\textsc{Stick}^{\mathsf{fix}}_{\mathsf{A}}$ and $\textsc{Stick}^{\mathsf{fix}}_{\mathsf{AB}}$. We describe the construction for $\textsc{Stick}^{\mathsf{fix}}_{\mathsf{A}}$ and afterwards consider the small differences to $\textsc{Stick}^{\mathsf{fix}}_{\mathsf{AB}}$. We reduce from Monotone 3-SAT, which is still NP-complete [Li97].

Monotone 3-SAT is a special version of the satisfiability problem. As in 3-SAT, one is given a Boolean formula Φ in conjunctive normal form (CNF) where each clause contains at most three literals – in our case exactly three distinct literals. It is called *monotone* because in each clause, the literals are either all positive or all negative. The task is to decide whether there is an assignment of Boolean values (true or false) to the variables such that Φ is true. In other words, the task is to decide whether Φ is satisfiable.

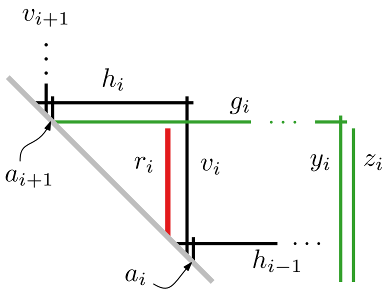
Theorem 3.8. $\textsc{Stick}^{\mathsf{fix}}_{\mathsf{A}}$ *is NP-complete.*

Proof. Recall that, as mentioned before, NP-membership follows from our linear program (see Theorem 3.12 in Section 3.3.3). More precisely, a total order σ of the vertices including σ_A as a subsequence serves as a small certificate. This total order is required as part of the input of our linear program.

To show NP-hardness, we describe a polynomial-time reduction from Monotone 3-SAT to $\textsc{Stick}^{\mathsf{fix}}_{\mathsf{A}}$. A schematization of our reduction is depicted in Figure 3.11. Given a Monotone 3-SAT instance Φ over variables $x_1, \dots, x_n$ and clauses $c_1, \dots, c_m$, we construct a *variable gadget* for each variable x_i (with $i \in \{1, \dots, n\}$) as depicted in Figure 3.10. Moreover, we construct a *clause gadget* for each clause c_j (with $j \in \{1, \dots, m\}$) as depicted in Figure 3.12.



(a) variable gadget set to false



(b) variable gadget set to true

Figure 3.10: Variable gadget in our reduction from Monotone 3-SAT to $\text{Stick}_\text{A}^\text{fix}$.

Variable Gadget. For each $i \in \{1, \ldots, n\}$, we construct a variable gadget of the variable v_i as follows. Inside a (black) *cage*, there is a vertical (red) stick r_i with length 1 and from the inside, a long horizontal (green) stick g_i leaves this cage. It is important that r_i and g_i are non-adjacent and r_i does not intersect the cage. We next describe how we enforce that the sticks arrange as in Figure 3.10.

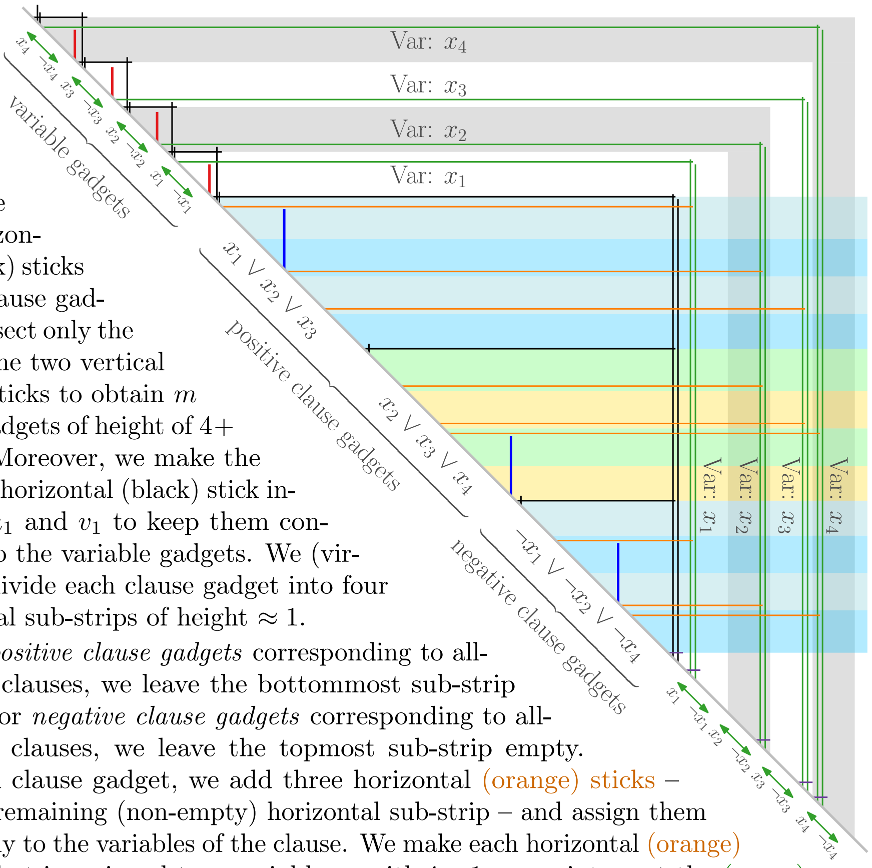
We prescribe the order σ_A of the vertical sticks as in Figure 3.10. The (black) sticks a_i and a_{i+1} get length $\varepsilon \ll 1$, which means that the horizontal (black) stick h_i intersects the two vertical (black) sticks v_{i+1} and a_{i+1} close to its foot point. Since we have prescribed $\sigma_A(a_{i+1}) < \sigma_A(r_i) < \sigma_A(v_i)$ and the stick r_i is inside the cage bounded by h_i and v_i, r_i enforces a minimum height of the cage. We give the sticks h_i and v_i each length $1 + 2\varepsilon$. Hence, h_i and v_i intersect close to their endpoints. Moreover, the stick r_i cannot be below h_{i-1} because a_i is shorter than r_i and intersects h_{i-1} to the right of r_i. This leaves the freedom of placing the stick g_i above or below the stick r_i but still g_i's foot point needs to be inside the cage formed by h_i and v_i because g_i intersects v_i, but neither v_{i-1} nor v_{i+1}.

We say that the variable x_i is set to false if the foot point of g_i is below the foot point of r_i, and to true otherwise. For each x_i, we add two long vertical (green) sticks y_i and z_i that we keep close together by using a short horizontal (violet) stick of length ε (see Figure 3.11 on the bottom right). We make stick g_i intersect stick y_i but not stick z_i. The three sticks g_i, y_i, and z_i get the same length ℓ_i. Hence, y_i and g_i intersect each other close to their endpoints as otherwise g_i would intersect z_i. We choose ℓ_1 sufficiently large such that the foot point of y_1 is always to the right of the clause gadgets (see Figure 3.11) and for each ℓ_i with $i \geq 2$, we set $\ell_i = \ell_{i-1} + 1 + 3\varepsilon$.

Now compare the endpoints of the stick g_i when x_i is set to false and when x_i is set to true relative to the (black) cage structure. When x_i is set to true, the endpoint of g_i is $1 \pm 2\varepsilon$ above and $1 \pm 2\varepsilon$ to the left compared to the case when x_i is set to false. Observe that, since g_i and y_i intersect each other close to their endpoints, this offset is also pushed to the sticks y_i and z_i and their foot points. Consequently, the position of the foot point of y_i (and z_i) differs by $1 \pm 2\varepsilon$ relative to the (black) frame structure depending on whether x_i is set to true or false. Our choice of ℓ_i allows this movement. In other words, no matter which truth value we assign to each variable x_i, there is a stick representation of the variable gadgets respecting σ_A.

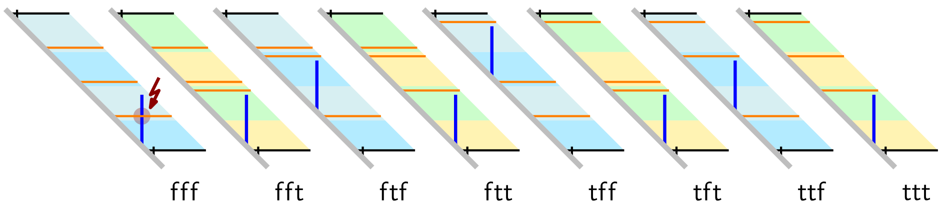
Clause Gadget. For each clause, we add a *clause gadget* (see Figure 3.12) as shown in Figure 3.11. It is a strip that is bounded by horizontal (black) sticks on its top and bottom. To fix the height of each strip, we introduce two vertical (black) sticks that we keep close together by a short horizontal (violet) stick of length ε (see Figure 3.11 on the bottom right).

Figure 3.11: Illustration of our reduction from Monotone 3-SAT to $\text{Stick}_{\text{A}}^{\text{fix}}$

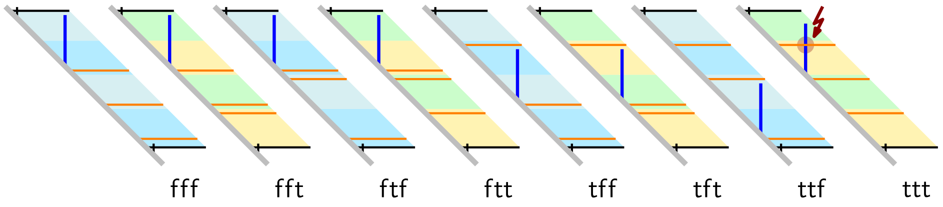


We make the horizontal (black) sticks of the clause gadget intersect only the first of the two vertical (black) sticks to obtain m clause gadgets of height of $4 + 2\varepsilon \pm \varepsilon$. Moreover, we make the topmost horizontal (black) stick intersect a_1 and v_1 to keep them connected to the variable gadgets. We (virtually) divide each clause gadget into four horizontal sub-strips of height ≈ 1.

For *positive clause gadgets* corresponding to all-positive clauses, we leave the bottommost sub-strip empty; for *negative clause gadgets* corresponding to all-negative clauses, we leave the topmost sub-strip empty. For each clause gadget, we add three horizontal (orange) sticks – one per remaining (non-empty) horizontal sub-strip – and assign them bijectively to the variables of the clause. We make each horizontal (orange) stick o that is assigned to a variable x_i with $i \in 1, \dots, n$ intersect the (green) sticks y_i and all y_j and z_j for $j < i$, but not z_i or y_k or z_k for any $k > i$. Thus, o intersects y_i close to o's endpoint. We choose the length of each horizontal stick o so that its foot point is at the bottom of its sub-strip if x_i is set to false and is at the top of its sub-strip if x_i is set to true. Within the positive and the negative clause gadgets, this gives us two times eight possible configurations of the orange sticks depending on the truth assignment of the three variables of the clause (see Figure 3.12).



positive clause gadget (empty sub-strip at the bottom)



negative clause gadget (empty sub-strip at the top)

Figure 3.12: Clause gadget in our reduction from MONOTONE 3-SAT to $\text{STICK}_{\text{A}}^{\text{fix}}$. Here, a clause gadget for each of the eight possible truth assignments of a MONOTONE 3-SAT clause is depicted. In particular, it shows that the isolated vertical (blue) stick fits inside the gadget if and only if the corresponding clause is satisfied. The combination of truth assignments is written below – e.g., tft means that the first variable is set to true, the second variable to false, and the third variable to true.

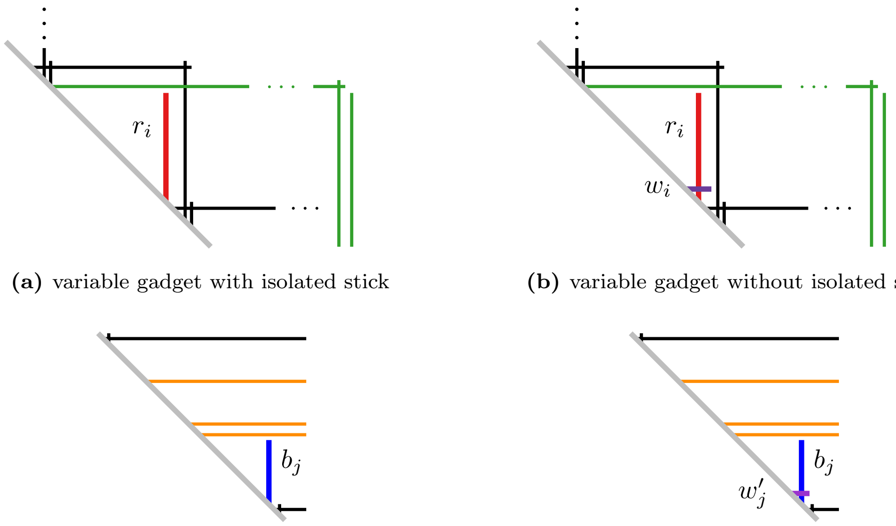
Within each clause gadget, we have a vertical (blue) stick b of length 2, which does not intersect any other stick. Each horizontal (black) stick that bounds a clause gadget intersects a short vertical (black) stick of length ε, which forces b to lie within its designated clause gadget due to the order σ_A.

Correctness. Clearly, if Φ is satisfiable, there is a stick representation of the $\text{STICK}_{\text{A}}^{\text{fix}}$ instance obtained from Φ by our reduction by placing the sticks as described before (see also Figure 3.11). In particular, each (blue) stick can be placed in one of the ways listed in Figure 3.12.

On the other hand, if there is a stick representation of the $\text{STICK}_{\text{A}}^{\text{fix}}$ instance obtained by our reduction, Φ is satisfiable. As argued before, the shape of the (black) frame structure of all gadgets is fixed by the choice of the adjacencies and lengths in the graph and σ_A. The only flexibility is, for each $i \in \{1, \ldots n\}$, whether stick g_i has its foot point above or below stick r_i. This enforces one of eight distinct configurations per clause gadget. As depicted in Figure 3.12, precisely the configurations that correspond to satisfying truth assignments are realizable. Thus, we can read a satisfying truth assignment of Φ from the variable gadgets.

Clearly, our reduction can be implemented in polynomial time. □

In our reduction, we enforce an order of the horizontal sticks. So, prescribing σ_B makes it even easier to enforce this structure. Hence, we can use exactly the same reduction for $\text{STICK}_{\text{AB}}^{\text{fix}}$.



(a) variable gadget with isolated stick

(b) variable gadget without isolated stick

(c) clause gadget with isolated stick

(d) clause gadget without isolated stick

Figure 3.13: We add a short horizontal (violet) stick w_i (or w'_j) that intersects stick r_i (or stick b_j) to avoid isolated sticks in a variable (or clause) gadget for variable x_i (or clause C_j).

Corollary 3.9. STICK$^{\mathsf{fix}}_{\mathsf{AB}}$ *(with isolated vertices in A or B) is* NP*-complete.*

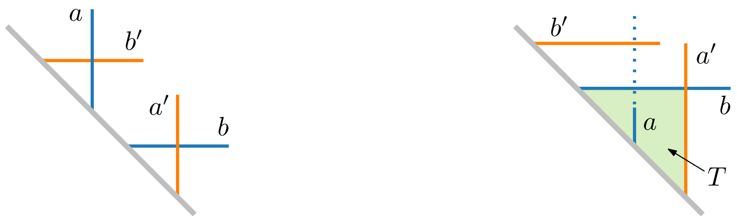
Proof. Given a MONOTONE 3-SAT instance Φ, consider the construction described in the proof of Theorem 3.8. We use the same graph, the same stick lengths and the same ordering σ_A of the vertical sticks. Now, we additionally prescribe the order of the remaining horizontal sticks as depicted in Figure 3.11 via σ_B. This defines an instance of STICK$^{\mathsf{fix}}_{\mathsf{AB}}$.

Clearly, the ordering of the horizontal sticks σ_B neither affects the placement of the isolated vertical (red) sticks inside a variable gadget nor does it affect the placement of the isolated vertical (blue) sticks inside a clause gadget. Moreover, there was only one possible ordering of the horizontal sticks in the construction described in the proof of Theorem 3.8. Thus, its correctness proof applies here as well. □

The reduction we described before uses isolated vertices inside the variable and the clause gadgets. In the case of STICK$^{\mathsf{fix}}_{\mathsf{AB}}$, this is indeed necessary to show NP-hardness. This is not true for STICK$^{\mathsf{fix}}_{\mathsf{A}}$, which remains NP-hard (and hence is NP-complete due to our linear program; see Section 3.3.3) even without isolated sticks.

Corollary 3.10. STICK$^{\mathsf{fix}}_{\mathsf{A}}$ *without isolated vertices is* NP*-complete.*

Proof. We use the same reduction as in the proof of Theorem 3.8, but we additionally add, for each isolated stick, a short (violet) stick of length $\varepsilon \ll 1$ that only intersects the isolated stick; see Figure 3.13. In each variable gadget, for the isolated vertical (red) stick r_i, we add a short horizontal (violet) stick w_i of length ε. Similarly, in



(a) a comes before b (valid representation) **(b)** a comes after b (no representation)

Figure 3.14: Trying to represent a subgraph of the edges ab' and $a'b$ while respecting σ_A and σ_B.

each clause gadget, for the isolated vertical (blue) stick b_j, we add a short horizontal (violet) stick w'_j of length ε. After these additions, no isolated sticks remain.

Observe that, for any placement of the isolated sticks inside their gadgets in the proof of Theorem 3.8, we can add the new horizontal (violet) stick since it has length only ε. Moreover, since these new sticks are horizontal, we do not get any new ordering constraints in the version $\textsc{Stick}_{\mathsf{A}}^{\mathsf{fix}}$.

We, therefore, conclude that the rest of the proof still holds true. □

3.3.3 $\textsc{Stick}_{\mathsf{AB}}^{\mathsf{fix}}$ without Isolated Vertices

In this section, we constructively show that $\textsc{Stick}^{\mathsf{fix}}$ is efficiently solvable if we are given a total order of the vertices in $A \cup B$ on the ground line.

First, we observe that if there is a stick representation for an instance of $\textsc{Stick}_{\mathsf{AB}}$ (and consequently also $\textsc{Stick}_{\mathsf{AB}}^{\mathsf{fix}}$), the combinatorial order of the sticks on the ground line is the same for any stick representation except for the isolated vertices, which we formalize in the following lemma. The proof also follows implicitly from the proof of Theorem 3.3.

Lemma 3.11. *In all stick representations of an instance of $\textsc{Stick}_{\mathsf{AB}}$, the order of the vertices $A \cup B$ on the ground line is the same after removing all isolated vertices. This order can be found in $\mathcal{O}(|E|)$ time.*

Proof. Assume there are stick representations Γ_1 and Γ_2 of the same instance of $\textsc{Stick}_{\mathsf{AB}}$ without isolated vertices that have different combinatorial arrangements on the ground line. Without loss of generality, there is a pair of sticks $(a, b) \in A \times B$ such that in Γ_1, a comes before b, while in Γ_2, a comes after b (see Figure 3.14). Clearly, a and b cannot be adjacent. Since a is not isolated, there is a stick b' that is adjacent to a and comes before b. Similarly, there is a stick a' that is adjacent to b and comes after a. In Γ_2, stick b, stick a', and the ground line define a triangular region T (see Figure 3.14b), which completely contains a since a occurs between b and a', but is adjacent to neither of them. However, b' is outside of T as it comes before b. This contradicts b and a' being adjacent. The unique order can be determined in $\mathcal{O}(|E|)$ time as described in Section 3.2. □

Second, we describe a linear program to compute a stick representation given a total order of the vertices on the ground line. We are given an instance of $\textsc{Stick}^{\mathsf{fix}}$ and a total order $v_1, \dots, v_n$ of the vertices with stick lengths $\ell_1, \dots, \ell_n$, where $n = |A| + |B|$. We create a system of difference constraints, that is, a linear program $Ax \le b$ where each constraint is a simple linear inequality of the form $x_j - x_i \le b_k$ with n variables and $m \le 3n - 1$ constraints. Such a system can be modeled as a weighted graph with a vertex per variable x_i and a directed edge (x_i, x_j) with weight b_k per constraint. The system has a solution if and only if there is no directed cycle of negative weight [Pra77, CLRS22]. In this case, a solution can be found in $\mathcal{O}(nm)$ time using the Bellman–Ford algorithm.

In the following, we describe how to construct such a linear program for $\textsc{Stick}^{\mathsf{fix}}$. For each stick v_i, we create a variable x_i that corresponds to the x-coordinate of v_i's foot point on the ground line, with $x_1 = 0$. To ensure the unique order, we add $n - 1$ constraints $x_i - x_{i+1} \le -\varepsilon$ for some suitably small ε and $i = 1, \dots, n - 1$.

Let $v_i \in A$ and $v_j \in B$. If $(v_i, v_j) \in E$, then the corresponding sticks have to intersect, which they do if and only if $x_i - x_j \le \min\{\ell_i, \ell_j\}$. If $j < i$ and $(v_i, v_j) \notin E$, then the corresponding sticks must not intersect, so we require $x_i - x_j > \min\{\ell_i, \ell_j\}$ or equivalently $x_i - x_j \ge \min\{\ell_i, \ell_j\} + \varepsilon$. This easily gives a system of difference constraints with $\mathcal{O}(n^2)$ constraints. We argue that a linear number suffices.

Let $v_i \in A$. Let j be the smallest j such that $(v_i, v_j) \in E$ and $\ell_j \ge \ell_i$. We add a constraint $x_i - x_j \le \ell_i$. Further, let k be the largest k such that $(v_i, v_k) \notin E$ and $\ell_k \ge \ell_i$. We add a constraint $x_i - x_k > \ell_i \Leftrightarrow x_k - x_i \le -\ell_i - \varepsilon$. Symmetrically, let $v_i \in B$. Let j be the largest j such that $(v_j, v_i) \in E$ and $\ell_j > \ell_i$. We add a constraint $x_j - x_i \le \ell_i$. Further, let k be the smallest k such that $(v_k, v_i) \notin E$ and $\ell_k > \ell_i$. We add a constraint $x_k - x_i > \ell_i \Leftrightarrow x_i - x_k \le -\ell_i - \varepsilon$.

We now argue that these constraints are sufficient to ensure that G is represented by a solution of the system. Let $v_i \in A$ and $v_j \in B$. If $i < j$, then the corresponding sticks cannot intersect, which is ensured by the fixed order. So assume that $j < i$. If $\ell_j \ge \ell_i$ and $(v_i, v_j) \in E$, then we either have the constraint $x_i - x_j \le \ell_i$, or we have a constraint $x_i - x_k \le \ell_i$ with $k < j < i$; together with the order constraints, this ensures that $x_i - x_j \le x_i - x_k \le \ell_i$. If $\ell_j \ge \ell_i$ and $(v_i, v_j) \notin E$, then we either have the constraint $x_j - x_i \le -\ell_i - \varepsilon$, or we have a constraint $x_k - x_i \le -\ell_i - \varepsilon$ with $j < k < i$; together with the order constraints, this ensures that $x_j - x_i \le x_k - x_i \le -\ell_i - \varepsilon$. Symmetrically, the constraints are also sufficient for $\ell_j < \ell_i$. We obtain a system of difference constraints with n variables and at most $3n - 1$ constraints proving Theorem 3.12.

Theorem 3.12. $\textsc{Stick}^{\mathsf{fix}}$ *can be solved in* $\mathcal{O}((|A| + |B|)^2)$ *time if we are given a total order of the vertices.*

By Lemma 3.11, there is at most one realizable order of vertices for a $\textsc{Stick}^{\mathsf{fix}}_{\mathsf{AB}}$ instance without isolated vertices, which can be found in linear time and proves Corollary 3.13 based on Theorem 3.12.

Corollary 3.13. $\textsc{Stick}^{\mathsf{fix}}_{\mathsf{AB}}$ *can be solved in* $\mathcal{O}((|A| + |B|)^2)$ *time if there are no isolated vertices.*

3.4 Concluding Remarks and Open Problems

In this chapter, we have shown that $\textsc{Stick}^{\mathsf{fix}}$ is NP-complete even if the sticks have only three different lengths, while $\textsc{Stick}^{\mathsf{fix}}$ for unit-length sticks is solvable in linear time. But what is the computational complexity of $\textsc{Stick}^{\mathsf{fix}}$ for sticks with one of *two* lengths? Also, the three different lengths in our proof depend on the number of sticks. Is $\textsc{Stick}^{\mathsf{fix}}$ still NP-complete if the fixed lengths are bounded?

We have shown that $\textsc{Stick}^{\mathsf{fix}}_{\mathsf{AB}}$ is NP-complete if there are isolated vertices (in at least one of the bipartitions). In our NP-hardness reduction we use a linear number of isolated vertices. Clearly, $\textsc{Stick}^{\mathsf{fix}}_{\mathsf{AB}}$ is in XP in the number n_{isolated} of isolated vertices. An XP-algorithm could first compute the unique ordering of the non-isolated vertices and then try to insert each isolated vertex at each possible position in the permutation brute-force. However, the question remains open whether $\textsc{Stick}^{\mathsf{fix}}_{\mathsf{AB}}$ is fixed-parameter tractable (FPT) in n_{isolated}.[8]

[8] This question has been asked by Paweł Rzążewski at the 27th International Symposium on Graph Drawing and Network Visualization 2019 (GD'19) in Prague.

Chapter 4

Lower Bounds on the Segment Number of Some Planar Graph Classes

In the following two chapters, we investigate planar graph drawings with low visual complexity. Here, we investigate the number of line segments required for a planar straight-line drawing. In the next chapter, we consider the number of distinct slopes.

4.1 Introduction

One of the main goals in graph drawing is to generate clear drawings. Depending on the particular use case, we may request that a graph drawing has specific properties and use quality measures for evaluation. Classic examples are few edge crossings, small drawing area, neighborhood preservation, or low stress of a drawing [DETT99].

Schulz [Sch15] proposed the *visual complexity* as a quality measure, which is determined by the number of different geometric primitives used for the drawing. Kindermann, Meulemans, and Schulz [KMS18] have experimentally shown that people without mathematical background tend to prefer drawings with low visual complexity and that for some tasks and some graphs it may be beneficial to use drawings with low visual complexity. The visual complexity of a graph drawing depends on the drawing style and the properties of the drawn graph. Typical examples of such geometric primitives are the number of line segments or circular arcs, distinct line segment slopes, symmetries, distinct x- and y-coordinates of the vertices, different whitespace widths, and many more.

Maybe the most basic of these measures is the number of line segments in a planar straight-line graph drawing. This number is known as the *segment number*, and it has first been introduced and studied by Dujmović, Eppstein, Suderman, and Wood [DESW07] in 2007. It is defined as follows. A *segment* in a planar straight-line drawing is a maximal set of edges that together form a line segment. Given a straight-line drawing Γ of a graph, the set of segments that Γ induces is unique. The cardinality of this set is the *segment number* of Γ. The *segment number* $\operatorname{seg}(G)$ of a planar graph G is the smallest segment number over all planar straight-line drawings of G.

Previous Work. Dujmović et al. [DESW07] pointed out two natural lower bounds for the segment number: (i) $\eta(G)/2$, where $\eta(G)$ is the number of odd-degree vertices in G, and (ii) the *slope number* $\operatorname{slope}(G)$ of G, which is defined as follows. The slope number $\operatorname{slope}(\Gamma)$ of a planar straight-line drawing Γ of G is the number of distinct

graph class	universal lower bound	existential upper bound	existential lower bound	universal upper bound
trees	$\frac{1}{2}\eta$ [DESW07]	—	—	$\frac{1}{2}\eta$ [DESW07]
cacti	$\frac{1}{2}\eta + \gamma$ L4.21	—	—	$\frac{1}{2}\eta + \gamma$ T4.22
maximal outerpaths	$\lfloor \frac{1}{2}n \rfloor + 2$ T4.6	$\lfloor \frac{1}{2}n \rfloor + 2$ P4.9	n [DESW07]	n [DESW07]
maximal outerplanar	$\frac{1}{5}n + \frac{7}{5}$ T4.12	$\frac{5}{13}n + \frac{24}{13}$ P4.13	n [DESW07]	n [DESW07]
2-trees	$\frac{1}{5}n + \frac{7}{5}$ T4.12	$\frac{5}{13}n + \frac{24}{13}$ P4.13	$\frac{3}{2}n - 2$ [DESW07]	$\frac{3}{2}n$ [DESW07]
planar 3-trees	$n + 4$ T4.14	$n + 7$ P4.15	$\frac{3}{2}n$ P4.16	$2n - 2$ [DESW07]
planar 3-connected	$\sqrt{2n}$ [DESW07]	$\mathcal{O}(\sqrt{n})$ [DESW07]	$2n - 6$ [DESW07]	$\frac{5}{2}n - 3$ [DESW07]
planar 3-conn. 3-regular	$\frac{1}{2}n + 3$ [DESW07]	—	—	$\frac{1}{2}n + 3$ [MNBR13, IMS17]
planar 3-conn. 4-regular	$\Omega(\sqrt{n})$ [GKK+22]	$\mathcal{O}(\sqrt{n})$ [GKK+22]	n [GKK+22]	$n + 3$ [GKK+22]
4-conn. triangulations	$\Omega(\sqrt{n})$ [DESW07]	$\mathcal{O}(\sqrt{n})$ [DESW07]	$2n - 6$ [DESW07]	$\frac{9}{4}(n - 1)$ [DM19]
triangulations	$\Omega(\sqrt{n})$ [DESW07]	$\mathcal{O}(\sqrt{n})$ [DESW07]	$2n - 2$ [DESW07]	$\frac{7}{3}n - \frac{10}{3}$ [DM19]
planar connected	1	1	$2n - 2$ [DESW07]	$\frac{8}{3}n - \frac{14}{3}$ [DM19, KMSS19]

Table 4.1: Universal and existential lower and upper bounds on the segment number for subclasses of planar graphs. By *existential upper bound* we mean an upper bound for the universal lower bound. Such a bound is provided by the segment number of a specific graph family within the given graph class. Here, n is the number of vertices, η is the number of odd-degree vertices, and $\gamma = 3c_0 + 2c_1 + c_2$, where c_i is the number of simple cycles with exactly i cut vertices. We use "—" to indicate that universal lower bound and the universal upper bound agree for a specific graph class. The corresponding algorithms are thus optimal. Blue entries mark the results presented in this chapter. They refer to theorems (T), propositions (P), and lemmas (L).

slopes used by the straight-line edges in Γ. Then slope(G) is the minimum of slope(Γ) over all planar straight-line drawings Γ of G. Dujmović et al. also showed that any tree T admits a drawing with $\text{seg}(T) = \eta(T)/2$ segments and $\text{slope}(T) = \Delta(T)/2$ slopes, where $\Delta(T)$ is the maximum degree of any vertex in T. For other graph classes, they show bounds that depend on the number of vertices n. Namely, every maximal outerplanar graph G admits an outerplanar straight-line drawing with at most n segments and this bound is tight. Further, they also give (asymptotically) worst-case optimal algorithms for drawing 2-trees with $3n/2$ segments and drawing plane 3-trees with $2n-2$ segments. Finally, they show that every triconnected planar graph can be drawn using at most $5n/2-3$ segments.

For the special cases of triangulations and 4-connected triangulations, Durocher and Mondal [DM19] improved the upper bound of Dujmović et al. to $(7n-10)/3$ and $(9n-9)/4$, respectively. The former bound implies a bound of $(16n-3m-28)/3$ for arbitrary planar graphs with n vertices and m edges. Kindermann, Mchedlidze, Schneck, and Symvonis [KMSS19] observed that this implies that $\text{seg}(G) \leq (8n-14)/3$ for any planar graph G: if $m > (8n-14)/3$ this follows from the bound, otherwise any straight-line drawing of G is good enough since every edge could have its own segment. Constructive linear-time algorithms that compute the segment number of series-parallel graphs of maximum degree 3 and of maximal outerpaths were given by Samee, Alam, Adnan, and Rahman [SAAR08] and by Adnan [Adn08], respectively. Mondal, Nishat, Biswas, and Rahman [MNBR13] and Igamberdiev, Meulemans, and Schulz [IMS17] proved that for every cubic triconnected planar graph G (except K_4) $\text{seg}(G) = n/2+3$. If instead of a cubic, we consider a 4-regular triconnected planar graph G, Goeßmann et al. [GKK$^+$22] showed that $\text{seg}(G) \leq n+3$ (note that there are $2n$ edges) and they gave a universal lower bound of $\text{seg}(G) \in \Omega(\sqrt{n})$.

Related Work. Concerning the computational complexity, Durocher, Mondal, Nishat, and Whitesides [DMNW13] showed that the segment number of a graph is NP-hard to compute, even if in the resulting planar drawing all faces need to be convex. Okamoto, Ravsky, and Wolff [ORW19] introduced and investigated new variants of the segment number: for planar graphs in 2D they allowed bends, and for arbitrary graphs, they considered crossing-free straight-line drawings in 3D and straight-line drawings with crossings in 2D. They proved that all segment number variants (including the "original" segment number) are $\exists\mathbb{R}$-complete to compute, and they gave upper and existential lower bounds for the segment number variants of cubic graphs.

So far, the listed results did not restrict the drawing area. In practice, a drawing with a small number of segments but a large drawing area might not be useful at all. Hültenschmidt, Kindermann, Meulemans, and Schulz [HKMS18] showed that trees, maximal outerplanar graphs, and planar 3-trees admit drawings on a grid of polynomial size if we allow $3n/4$, and $3n/2$, and $(8n-17)/3$ segments, respectively. For trees with the minimum number of segments (i.e., $\eta(T)/2$ segments), they could show that a quasipolynomial-size grid is always sufficient and left it as an open question whether this can be reduced to a polynomial-size grid. Kindermann et al. [KMSS19] improved these bounds by reducing the grid size of trees with $\leq 3n/4$ segments and extending the result for maximal outerplanar graphs to (biconnected) outerplanar graphs. Moreover, they introduced bounds on the grid size for triangulations and (3- and 4-connected) planar graphs.

Instead of line segments, we can also consider graph drawings where the edges are drawn with circular arcs. Such drawings are also known as *Lombardi drawings*, which are sometimes considered more visually pleasing [PHNK12, XRPH12]. For a graph G, the *arc number* $\operatorname{arc}(G)$ is the smallest number of circular arcs in any planar circular-arc drawings of G, where besides partial circles also closed circles are allowed. Schulz [Sch15] introduced the arc number in 2015. He also gave algorithms for drawing series-parallel graphs, planar 3-trees, and triconnected planar graphs with $m/2+1$, $11m/18+3$, $2m/3$ circular arcs, respectively, where m is the number of edges in the given graph. For trees, he exploited the flexibility of circular arcs to reduce the drawing area to polynomial size. In particular, Schulz showed that circular-arc drawings are an improvement over straight-line drawings not only in terms of visual complexity but also in terms of area consumption. Later Hültenschmidt et al. [HKMS18] added universal upper bounds on the arc number for (4-connected) triangulations, and (4-connected) planar graphs. For instance, the bound for any triangulation G is $\operatorname{arc}(G) \leq 5n/3$, whereas there are triangulations that require $2n-2$ segments. Chaplick, Förster, Kryven, and Wolff [CFKW20] considered circular arc drawings in combination with right-angle crossings – another criterion for better readability of a graph.

Chaplick, Fleszar, Lipp, Ravsky, Verbitsky, and Wolff [CFL$^+$17, CFL$^+$20] investigated the *line cover number* and the *plane cover number* as measures of the visual complexity. It is the number of lines (in 2D and 3D) or planes (in 3D) needed to cover crossing-free straight-line drawings of graphs, respectively. Clearly, the line cover number in 2D of a graph G is a lower bound for the segment number of G. These numbers are not identical because two non-connected line segments on the same line count as two segments for the segment number, but only as one line for the line cover number. Similarly, Kryven et al. [KRW19] considered circle covers and spherical covers.

Contribution. In this chapter, we focus on *universal lower bounds* on the segment number for some classes of planar graphs. We call it a universal lower bound because any graph G of the considered graph class requires this number of segments. We prove the first linear universal lower bounds for maximal outerpaths ($\lfloor n/2 \rfloor + 2$; see Section 4.2), maximal outerplanar graphs as well as 2-trees (both $(n+7)/5$; see Section 4.3), and planar 3-trees ($n+4$; see Section 4.4). This makes the corresponding algorithms of Dujmović et al. [DESW07] constant-factor approximation algorithms. Our universal lower bound for maximal outerpaths provides a lower bound on the solution returned by Adnan's algorithm [Adn08], which computes the segment number of maximal outerpaths. Moreover, our bound is tight and can be generalized to circular arcs and arrangements of pseudo-k-arcs (defined below). For planar 3-trees, our bound is tight up to the additive constant. We also give a simple optimal algorithm for cactus graphs (see Section 4.6), generalizing the result of Dujmović et al. for trees.

Circular-arc drawings are a natural generalization of straight-line drawings. Therefore, it is interesting to investigate how much of an improvement the former offer over the latter in terms of visual complexity. Specifically, we discuss bounds on the ratio $\mathrm{seg}(G)/\mathrm{arc}(G)$ between the segment and the arc number of a graph; see Section 4.5. Known and new results are listed in Table 4.1.

Definitions and Notation. In a straight-line drawing Γ of a graph G, each segment terminates at two vertices. Let S be a segment in Γ, and let v be an endpoint of S. Geometrically speaking, we could extend S at v into a face f. We say that S has a *port* at v in f. We call v *open* if v has at least one port and *closed* otherwise. Let $\mathrm{port}(\Gamma)$ be the number of ports in Γ, and let $\mathrm{port}(G)$ be the minimum number of ports over all straight-line drawings of G. Observe that, for any planar graph G, it holds that $\mathrm{seg}(G) = \mathrm{port}(G)/2$. Hence, in a drawing of G, counting segments is equivalent to counting ports.

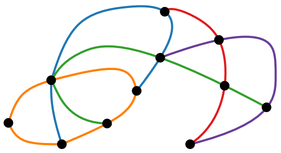
4.2 Maximal Outerpaths

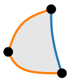
In this section, we prove the universal lower bounds of $\lceil \frac{n}{2} \rceil + 2$ for the segment number and of $\lceil \frac{2n+1}{7} \rceil$ for the arc number of maximal outerpaths, where n denotes the number of vertices. To this end, we generalize arrangements of segments and arcs to arrangements of pseudo-k-arcs (defined below) and then give a universal lower bound for the number of pseudo-k-arcs in drawings of maximal outerpaths based on an elaborate charging scheme. We first need a few more definitions and observations before we can describe and then analyze the charging scheme; at the end of this section, we also consider the tightness of these bounds.

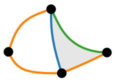
Pseudo-k-arc Arrangements. An arrangement of *pseudo-k-arcs* is a set of curves in the plane such that any two of the curves intersect at most k times; two curves touching counts as two intersections. While we forbid self-intersections, we allow a pseudo-k-arc to be closed for $k \geq 2$. The cases $k = 1$ and $k = 2$ are of special interest since they correspond to pseudosegments and pseudo-circular arcs, which in turn are generalizations of segments and arcs. We adopt these generalizations from the well-known concepts of pseudoline and pseudocircle arrangements, which generalize arrangements of lines and circles [FG17, RFO^{+}24]. We assume that we work with topological descriptions of these arrangements.

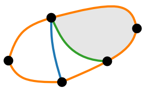
We define a drawing of a graph on a pseudo-k-arc arrangement in the same way as a drawing on an arrangement of segments or arcs: (i) the vertices are drawn on all endpoints, all intersection points, and some inner points of the pseudo-k-arcs, and (ii) there is an edge between two vertices if and only if there is (a section of) a pseudo-k-arc drawn between them not containing another vertex. As before, we do not allow parallel edges.

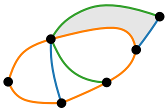
Recall that a maximal outerpath is a 2-tree. Hence a sequence $v_1, v_2, \ldots, v_n$ of the vertices of a maximal outerpath G is a stacking order of G if for each i, the graph G_i induced by the vertices $v_1, v_2, \ldots, v_i$ is a maximal outerpath. In the


(a) Complete drawing.


(b) Initially, the orange and blue arc are active.


(c) The green arc appears; all three arcs are active.


(d) We close the orange arc; it does not have an endpoint at the new face and, hence, becomes inactive.


(e) We extend the green arc and the other end of the blue arc appears, which we do not count as a new arc.

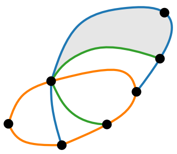

(f) We close the blue arc, which becomes inactive now; only the green arc remains active.

Figure 4.1: Maximal outerpath drawing on five pseudo-3-arcs considered sequentially according to its stacking order.

following, we consider an n-vertex maximal outerpath G along a stacking order of G such that all G_i, $i \in \{3, \ldots, n\}$, share a degree-two vertex of G; in other words, we build G along its dual path. Given a drawing Γ of G, we denote the sub-drawings of $G_3, G_4, \ldots, G_n$ within Γ by $\Gamma_3, \Gamma_4, \ldots, \Gamma_n$, respectively. We sometimes use temporal words to refer to the incremental changes from Γ_3 to $\Gamma_n = \Gamma$. A pseudo-k-arc α is *incident* to a face f if α contains an edge incident to a vertex of f. We say that α is *active* in Γ_i (for $i \in \{3, \ldots, n\}$) if α is incident to the face f that has been added in Γ_i (with respect to Γ_{i-1}) and one of its endpoints in Γ_i is at a vertex of f. Furthermore, we say an edge e *appears* in a drawing Γ_i if e is drawn with both endpoints in Γ_i but not in Γ_{i-1}. Similarly, a pseudo-k-arc α *appears* in Γ_i if at least one edge represented by α appears in Γ_i but no edge represented by α appears in a Γ_j with $j < i$. Therefore, we can speak of edges and pseudo-k-arcs appearing *before*, *at the same time*, or *after* other edges or pseudo-k-arcs. If it is clear from the context that we mean a pseudo-k-arc, we sometimes just call it an *arc* for short. For an example, see Figure 4.1.

We say a pseudo-k-arc is *long* if it contains at least $k+1$ inner edges; otherwise it is *short*. Note that when an arc α appears, it is always short. We say α *becomes long* in Γ_i if the $(k+1)$-th inner edge e_{k+1} of α is an inner edge in Γ_i but not in Γ_{i-1}. This implies that e_{k+1} appears in Γ_{i-1}.

When it comes to counting pseudo-k-arcs in Γ, we let arc_k denote the number of pseudo-k-arcs, and we let arc_k^i ($\mathrm{arc}_k^{>i}$) denote the number of pseudo-k-arcs with i (resp. more than i) inner edges. The inner edges of a long arc α subdivide the outerpath into subgraphs $H_0, H_1, \ldots, H_\ell$ called *bays*; see Figure 4.2.

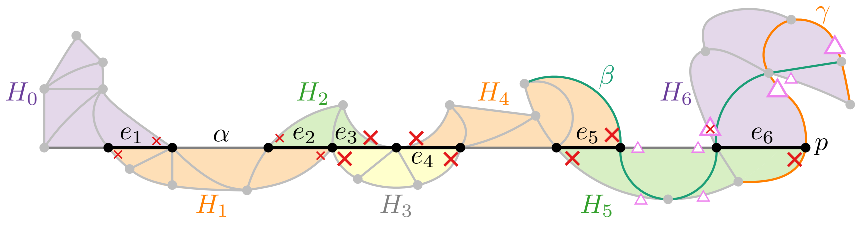


Figure 4.2: A maximal outerpath represented by a pseudo-2-arc arrangement. The inner edges $e_1, \ldots, e_6$ of arc α subdivide the outerpath into bays $H_0, \ldots, H_6$. The bay crossings of α and β are marked by red crosses and violet triangles, respectively. For the bay crossings in C, which are relevant for our charging scheme, we use larger symbols.

Initial Observations. Let us first describe a few observations on which our charging scheme is build on.

Observation 4.1. *Let e be an edge of G and let $i \in \{3, \ldots, n\}$. If e is an inner edge in Γ_i but no inner edge in Γ_{i-1}, then e is incident to all active pseudo-k-arcs of Γ_i.*

Note that in Figure 4.2, β becomes long after the first long arc α has become inactive. In fact, this is true for all long pseudo-k-arcs, leading to the following lemma.

Lemma 4.2. *For any $i \in \{3, \ldots, n\}$, Γ_i contains at most one active long pseudo-k-arc. Long pseudo-k-arcs appear in the same order as they become long.*

Proof. Let α and β be any two pseudo-k-arcs such that α appears before β. Suppose that in Γ_i, β is long and active and α is active and either also long or becomes long in some Γ_j with $j > i$. For β to become long, β must have $k+1$ inner edges, while α remains active. Let $H_0, H_1, \ldots, H_\ell$ be the bays of β, i.e. the subgraphs into which the inner edges of β subdivide the complete outerpath drawing Γ. Now for α to leave H_0, α needs either to enter H_1 (which requires an intersection between α and β) or to enter H_2 (which requires a tangential point of α at β and is counted as two intersections). For α to be active when β is long, α needs to reach H_{k+1} (or more general, some $H_{k'}$ with $k' \geq k+1$). This however, requires at least $k+1$ intersection points between α and β (see Observation 4.1). This is a contradiction to the definition of pseudo-k-arcs and α cannot be active any more when β becomes long. □

Charging Scheme. The high-level idea of the charging scheme is to assign all $n-3$ inner edges of G to pseudo-k-arcs in two rounds: In round 1, short pseudo-k-arcs pay fully and long pseudo-k-arcs pay partially for themselves; in round 2, long pseudo-k-arcs give their remaining charge to other pseudo-k-arcs in their vicinity. However, instead of constructing an explicit assignment in round 2, we (indirectly) establish a relation between the number of inner edges that have been excluded in round 1 and the "capacities" of other pseudo-k-arcs. To this end, we develop several bounds with a series of technical lemmas that ultimately allow us to derive the universal lower bounds.

We start with describing *round 1*. Let I denote the set of inner edges of long pseudo-k-arcs starting at the $(k+1)$-th inner edge (as for the first k inner edges an arc is still short). We assign each of the $n-3$ inner edges of G except for the edges in I to their own pseudo-k-arc. The number of these inner edges can be expressed as

$$\begin{aligned}(n-3) - |I| &= k \operatorname{arc}_k^{\geq k} + (k-1) \operatorname{arc}_k^{k-1} + (k-2) \operatorname{arc}_k^{k-2} + \dots + \operatorname{arc}_k^1 \\ &= k \operatorname{arc}_k - \textstyle\sum_{i=0}^{k} (k-i) \operatorname{arc}_k^i \ .\end{aligned} \tag{4.1}$$

Now we describe *round 2*, where we charge the inner edges in I to specific crossings, which in turn we can charge to pseudo-k-arcs. A *crossing* is a triple (α, β, p) that consists of two pseudo-k-arcs α and β and a point p at which α and β intersect. We consider specific crossings that involve a long arc and we call them *bay crossings*. Next, we define them such that for each long pseudo-k-arc α with ℓ inner edges $(\ell > k)$, there are 2ℓ bay crossings $(\alpha, *, *)$ where $*$ acts as wildcard. For each bay $H \in \{H_1, \dots, H_{\ell-1}\}$, we have two bay crossings: At each of the two vertices of H that have degree 2 within H, there is a crossing of α with another pseudo-k-arc; see the red crosses in Figure 4.2. Since these two vertices are distinct for each pair of consecutive bays, their bay crossings are distinct as well. Note that a tangential point may be shared by some H_j and H_{j+2} (for $j \in \{1, \dots, \ell-3\}$); see, e.g., H_2 and H_4 in Figure 4.2. However, we still have distinct bay crossings for H_j and H_{j+2} since a tangential point counts for two crossings. For $H \in \{H_0, H_\ell\}$, we define one bay crossing as follows. Consider the two crossings of α at the two vertices of the inner edge e_1 (resp. e_ℓ). One of these vertices is the degree-2 vertex of H_1 (resp. $H_{\ell-1}$) and hence may be identical with a bay crossing of H_1 (resp. $H_{\ell-1}$). For example in Figure 4.2, the bay crossing (α, γ, p) of H_5 occurs as one of the considered crossings of H_6. The other one of the two considered crossings cannot be a bay crossing in a neighboring bay and this is our bay crossing of H_0 (resp. H_ℓ); see the red crosses at H_0 and H_6 in Figure 4.2.

In round 2, we charge the surplus inner edges of a long arc α to pseudo-k-arcs involved in the bay crossings with α. For each inner edge e of I, we have two distinct bay crossings incident to e – one towards the preceding and one towards the succeeding bay (e.g., in Figure 4.2 the two red crosses where e_3 bounds H_2 and H_3), which account for e. However, there is one exception: for the last inner edge e_ℓ of each long arc, we count only one bay crossing (for simplicity, we do not count the bay crossing of H_ℓ). The reason is that either the bay crossing of H_ℓ or the second bay crossing of $H_{\ell-1}$ may be at the same point and with the same short arc as the bay crossing accounting for the $(k+1)$-th inner edge of the next long arc (e.g., in Figure 4.2 the bay crossing of H_6 (red cross) with respect to the first long arc α matches a counted bay crossing (violet triangle) of the second long arc β). Let C be the set of the bay crossings that account for the surplus inner edges of long arcs. The bay crossings of $H_0, \dots, H_{k-1}$, and H_ℓ as well as one bay crossing of H_k are not included in C as the inner edges $e_1, \dots, e_k$ are not contained in I. Clearly, for every long arc, the number of bay crossings in C is two times its edges in I minus one (for the last inner edge). Overall, this means $|C| = 2|I| - \operatorname{arc}_k^{>k}$, where $\operatorname{arc}_k^{>k}$ is the number of long pseudo-k-arcs.

Analysis. Next, we give an upper bound for $|C|$ in terms of arc_k. The main argument we exploit is that, by definition, each pseudo-k-arc can participate in at most k crossings with the (current) long arc and, hence, also in at most k bay crossings with the (current) long arc. However, we need to be careful when one long pseudo-k-arc becomes inactive and a new pseudo-k-arc becomes long, i.e., we consider the transition between one long arc to a new long arc. We need to show that a (not necessarily long) pseudo-k-arc γ does not contribute k crossings in C with multiple long arcs.

For an arc γ, let C_γ be the set of bay crossings in C where γ is the short arc. Note that since $|C| = \sum_{\gamma\in\Gamma}|C_\gamma|$, we can obtain a bound on $|C|$ by bounding $|C_\gamma|$. First observe that the first long arc $\alpha^\star$ is never the short arc in a bay crossing of C; hence, $C_{\alpha^\star} = \emptyset$. Now let γ be an arc that is not the first long arc and, for some $t_\gamma \geq 0$, let $\alpha_1, \alpha_2, \ldots, \alpha_{t_\gamma}$, in the order of appearance, be the arcs that are long while γ is active and before γ becomes long (if it becomes long).

It follows from Observation 4.1 that each $\alpha \in \{\alpha_1, \ldots, \alpha_{t_\gamma}\}$ appears before γ appears: since γ is active while α is long, γ reaches the bay H_{k+1} of α or a later bay. If γ appeared before α, it would appear in bay H_0 of α. Then, to reach H_{k+1} or a later bay, γ needs $k+1$ or more crossings with α, which contradicts the definition of pseudo-k-arcs. (Imagine that an arc needs to "pay" one crossing with α to reach the next bay of α due to Observation 4.1. This is also true for a touching point, where an arc may "pay" two crossings to skip one bay.) Furthermore, due to Lemma 4.2, α_1 becomes long before α_2 becomes long, which in turn becomes long before α_3 becomes long, and so on.

Next we develop a bound on $|C_\gamma|$ by analyzing the distribution of the inner edges of $\{\alpha_1, \ldots, \alpha_{t_\gamma}\}$ with respect to γ and with respect to each other. To this end, we introduce, for each γ, the following sets and numbers, for which we give an example afterwards. These sets and numbers allow us to apply a series of technical arguments in Lemma 4.3 yielding our bound. For every $i \in \{1, \ldots, t_\gamma\}$, let ...

- A_i be the set of inner edges of α_i that become inner edges before γ appears. We define $a_i = |A_i|$.
- B_i be the set of inner edges of α_i that become inner edges after or at the same time as γ appears and while α_{i-1} is active. We define $B_1 = \emptyset$ and $b_i = |B_i|$.
- C_i be the set of inner edges of α_i that become inner edges after or at the same time as α_{i-1} (if existent) becomes inactive and before α_i becomes long. We define $c_i = |C_i|$.
- D_i be the set of inner edges of α_i that become inner edges after or at the same time as α_i becomes long and while γ is active. We define $d_i = |D_i|$.
- $D_i^\star$ be the set of inner edges in D_i (thus, $D_i^\star \subseteq D_i$) where γ is in both incident bay crossings the short arc. We define $d_i^\star = |D_i^\star|$.

For example, with respect to γ (orange) in Figure 4.2, there are two long arcs $\alpha_1 = \alpha$ (black) and $\alpha_2 = \beta$ (green). As α has five inner edges before γ appears, $a_1 = 5$. Then, $b_1 = 0$ by definition, $c_1 = 0$ because α is long already when γ appears, and

$d_1 = 1$ since e_6 is the only inner edge of α after γ has appeared. As β has two inner edges before γ appears, we get $a_2 = 2$. Furthermore, $b_2 = c_2 = 0$ because β had already $k = 2$ inner edges when γ appears, and $d_2 = 2$ as β has two inner edges after it has become long and while γ is active. Note that an inner edge of α_i may belong to different sets with respect to different γ.

Lemma 4.3. *For any pseudo-k-arc γ, the number of bay crossings in C where γ is the short arc is bounded by $|C_\gamma| \le k + t_\gamma - 1 + \sum_{i=1}^{t_\gamma - 1} d_i^\star$.*

Proof. We first collect a bunch of equations and inequalities in Equations (4.2) to (4.6). Afterwards, we combine them straightforwardly to establish this bound.

Let $i \in \{2, \dots, t_\gamma\}$. Since α_i has k inner edges before it is long, we know that

$$a_i + b_i + c_i = k\,. \tag{4.2}$$

(Note that Equation (4.2) does not apply to $i = 1$ since α_1 can be long before γ appears.)

Until α_{i-1} becomes inactive, there is a crossing between α_{i-1} and α_i for every inner edge of α_i. Of course, there are at most k such crossings, which implies

$$a_i + b_i + f_i \le k\,, \tag{4.3}$$

where $f_i = |F_i|$, and F_i is the set of crossings between α_i and α_{i-1} that are not incident to an edge from $A_i \cup B_i$. For example, in Figure 4.2 with respect to γ, $f_2 = 0$ because both crossing points between $\alpha_1 = \alpha$ and $\alpha_2 = \beta$ are incident to inner edges from A_2 (recall that A_2 contains inner edges of β before γ appears).

Furthermore, for every inner edge of α_i where $i \in \{1, \dots, t_\gamma - 1\}$, there is a crossing between α_i and α_{i+1} after γ has appeared. On the side of α_{i+1}, such a crossing is contained in F_{i+1}, or is incident to an inner edge from B_{i+1}, or is incident to the last inner edge from A_{i+1}. Therefore, we have

$$b_i + c_i + d_i \le b_{i+1} + f_{i+1} + 1\,. \tag{4.4}$$

After the appearance of γ, every α_i has for every inner edge at least one crossing with γ (for the ones in $D_i^\star$ even two). Hence, we have, for each $i \in \{1, \dots, t_\gamma\}$,

$$b_i + c_i + d_i + d_i^\star \le k\,. \tag{4.5}$$

We can partition the set of bay crossings C_γ into t_γ groups depending on the long arcs. For each $i \in \{1, \dots, t_\gamma\}$, we let C_γ^i denote the corresponding subset. Since each bay crossing in C_γ^i has α_i as its long arc and occurs at an endpoint of an edge in D_i (or at two endpoints of an edge in $D_i^\star$), we know that

$$|C_\gamma^i| \le d_i + d_i^\star\,. \tag{4.6}$$

Now we have all ingredients to bound $|C_\gamma|$. In the following, the numbers in brackets refer to the equations and inequalities that we use.

$$\begin{aligned}
|C_\gamma| = \sum_{i=1}^{t_\gamma} |C_\gamma^i| &\overset{(4.6)}{\leq} \sum_{i=1}^{t_\gamma} d_i + d_i^\star \\
&\overset{(4.4)}{\leq} d_{t_\gamma} + d_{t_\gamma}^\star + \sum_{i=1}^{t_\gamma - 1} (b_{i+1} + f_{i+1} + 1 - b_i - c_i + d_i^\star) \\
&= d_{t_\gamma} + d_{t_\gamma}^\star + b_{t_\gamma} + t_\gamma - 1 + \sum_{i=1}^{t_\gamma - 1} (f_{i+1} - c_i + d_i^\star) \\
&\overset{(4.3)}{\leq} d_{t_\gamma} + d_{t_\gamma}^\star + b_{t_\gamma} + t_\gamma - 1 + \sum_{i=1}^{t_\gamma - 1} (k - a_{i+1} - b_{i+1} - c_i + d_i^\star) \\
&\overset{(4.2)}{=} b_{t_\gamma} + c_{t_\gamma} + d_{t_\gamma} + d_{t_\gamma}^\star + t_\gamma - 1 - c_1 + \sum_{i=1}^{t_\gamma - 1} (k - k + d_i^\star) \\
&\overset{(4.5)}{\leq} k + t_\gamma - 1 + \sum_{i=1}^{t_\gamma - 1} d_i^\star
\end{aligned}$$

□

To obtain a bound for $|C|$, it is a natural approach to add up the bounds of $|C_\gamma|$ specified in Lemma 4.3 for every pseudo-k-arc γ. However, if we just add up these bounds, we may have a number that is above the actual $|C|$ because multiple arcs are active simultaneously and in our worst-case analysis we expect all of them to be involved in all possible bay crossings in C. To be more accurate, we introduce, for each arc γ, (a lower bound for) the over-counting o_γ, which we can safely subtract when we add up all $|C_\gamma|$ to get a bound for $|C|$. Thereafter, we provide examples where we over-count.

Note that incident to the last inner edge e_ℓ of a long arc, there is only one bay crossing in C. Hence, for all $i \in \{1, \ldots, t_\gamma - 1\}$, at most one of $\{\gamma, \alpha_{i+1}, \ldots, \alpha_{t_\gamma}\}$ contributes a bay crossing in C being incident to the last inner edge e_ℓ of α_i, however, all of them have a crossing with α_i at e_ℓ due to Observation 4.1. The same holds true for every inner edge e of α_i ($i \in \{1, \ldots, t_\gamma - 1\}$) where γ contributes two bay crossings: there is at least one more long arc, namely α_{i+1}, that does not contribute a bay crossing in C being incident to e, however, α_{i+1} has a crossing with α_i at e. Thus,

$$o_\gamma \geq (t_\gamma - 1) + \sum_{i=1}^{t_\gamma - 1} d_i^\star \,. \tag{4.7}$$

For example, consider the arcs β and γ in Figure 4.2. All $d_i^\star$ are 0, $t_\beta = 1$ because α is the only arc that is long while β is active and short, and $t_\gamma = 2$ because α and β are the long arcs while γ is active. Due to our bound, we know that $|C_\beta| \leq 2$ and

$|C_\gamma| \leq 3$. However, for both we have counted a bay crossing at p incident to e_6 of α, which is only possible for one of them. We know this when we consider γ, to which β is a succeeding long arc, resulting in $o_\gamma \geq 1$. We do not over-correct because we assume o_β to be 0. For an example with "large" over-counting see Figure 4.4a where $k = 1$. There, almost all arcs are long arcs with two inner edges and the short arc of all r bay crossings in C is contributed by only one arc (the top-boundary segment).

With Lemma 4.3 and Equation (4.7), we can now prove an upper bound on $|C|$.

Lemma 4.4. *The number of bay crossings in C is bounded by $|C| \leq k(\operatorname{arc}_k - 3) - 1$.*

Proof. Beside the over-counting o_γ described for every arc γ, we remark that the first $2k$ bay crossings of the first long arc $\alpha^\star$ and the last bay crossing of the last long arc are not counted in C and they cannot overlap with some other crossing in C. Hence, we can additionally subtract $o = 2k + 1$. In the following, $\mathcal{A}$ is the set of all pseudo-k-arcs in Γ.

$$\begin{aligned}
|C| = \sum_{\gamma\in\mathcal{A}\setminus\{\alpha^\star\}} |C_\gamma| &\overset{\text{Lem. 4.3}}{\leq} \sum_{\gamma\in\mathcal{A}\setminus\{\alpha^\star\}} \left(k + t_\gamma - 1 + \sum_{i=1}^{t_\gamma-1} d_i^\star \quad - o_\gamma\right) - o \\
&\overset{(4.7)}{\leq} \sum_{\gamma\in\mathcal{A}\setminus\{\alpha^\star\}} \left(k + t_\gamma - 1 + \sum_{i=1}^{t_\gamma-1} d_i^\star \quad - \left((t_\gamma - 1) + \sum_{i=1}^{t_\gamma-1} d_i^\star\right)\right) - 2k - 1 \\
&= \sum_{\gamma\in\mathcal{A}\setminus\{\alpha^\star\}} k \quad - 2k - 1 = k(\operatorname{arc}_k - 3) - 1
\end{aligned}$$

□

We employ Lemma 4.4 to bound the number of surplus inner edges of the long arcs as

$$\begin{aligned}
|I| = \frac{|C| + \operatorname{arc}_k^{>k}}{2} &\leq \frac{k(\operatorname{arc}_k - 3) - 1 + \operatorname{arc}_k - \sum_{i=0}^{k} \operatorname{arc}_k^i}{2} \\
&= \frac{k+1}{2}\operatorname{arc}_k - \frac{3k + 1 + \sum_{i=0}^{k} \operatorname{arc}_k^i}{2} .
\end{aligned} \tag{4.8}$$

Plugging Equation (4.8) into Equation (4.1), we obtain the following general formula, which gives a lower bound on the number of pseudo-k-arcs for any maximal outerpath relative to n and k.

$$\begin{aligned}
(n-3) - \frac{k+1}{2}\operatorname{arc}_k + \frac{3k + 1 + \sum_{i=0}^{k} \operatorname{arc}_k^i}{2} &\leq k \operatorname{arc}_k - \sum_{i=0}^{k} (k - i) \operatorname{arc}_k^i \\
\Leftrightarrow \quad \frac{3k+1}{2}\operatorname{arc}_k &\geq \frac{2(n-3) + 3k + 1 + \sum_{i=0}^{k} (2(k-i)\operatorname{arc}_k^i + \operatorname{arc}_k^i)}{2} \\
\Leftrightarrow \quad \operatorname{arc}_k &\geq \frac{2n + 3k - 5 + \sum_{i=0}^{k} (2k - 2i + 1)\operatorname{arc}_k^i}{3k+1}
\end{aligned} \tag{4.9}$$

In the following, we use Equation (4.9), which holds for general pseudo-k-arc arrangements, to obtain bounds for specific values of k – in particular for $k = 1$, which are pseudosegment arrangement, and for $k = 2$, which are pseudo-circular arc arrangements. Note that the unresolved variables arc_k^i, which describe the number of pseudo-k-arcs having i inner edges, may differ for specific values of k. We only prove non-zero lower bounds for $k = 1$, i.e., for pseudosegments.

Segments and Pseudosegments. In the following, we mainly plug $k = 1$ into Equation (4.9). To this end, we first determine lower bounds for arc_1^0 and arc_1^1 in Lemma 4.5. They improve just the constant summand in the universal lower bound of maximal outerpaths specified in Theorem 4.6. Since this bound holds for pseudosegment arrangements, this directly implies a lower bound on the (practically more interesting) segment number.

Lemma 4.5. *In any maximal outerpath drawing onto a pseudosegment arrangement (i.e., a pseudo-k-arc arrangement where $k = 1$) either $\mathrm{arc}_1^0 \geq 3$ or both $\mathrm{arc}_1^0 \geq 2$ and $\mathrm{arc}_1^1 \geq 3$.*

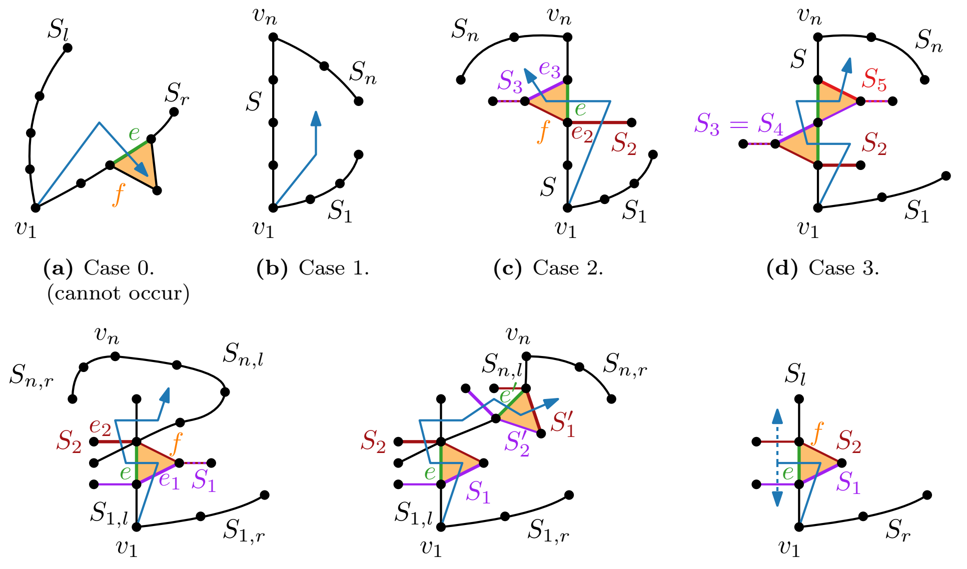
Proof. Consider v_1 and v_n, i.e., the first and the last vertex in the stacking order of G. Each of them lies on the endpoints of two pseudosegments. If they would lie on only one pseudosegment S, S would intersect the pseudosegment connecting the two neighbors of v_1 (or v_n) twice.

First, we show that v_1 and v_n have at least one incident pseudosegment with zero inner edges each (**Case 0**). Let v_1 be incident to the pseudosegments S_l and S_r and suppose both of them have at least one inner edge. Without loss of generality, the first inner edge e of S_r precedes the first inner edge of S_l when traversing the outerpath starting at v_1; see Figure 4.3a. The path of faces reaches the face f when passing over e. However, S_l is not incident to f and becomes inactive. (S_l cannot be incident to f because then S_l and S_r would intersect twice or v_1 would have degree > 2.) Therefore, S_l has zero inner edges. The same holds when traversing the outerpath backwards starting at v_n.

Using this property, we now can make the following case distinction of six cases.

Case 1: v_1 and v_n are incident to the same pseudosegment S having zero inner edges. Let the other pseudosegments being incident to v_1 and v_n be S_1 and S_n, respectively (clearly, they are distinct); see Figure 4.3b. This means that S is incident to all faces in the outerpath. So, if S_1 or S_n had an inner edge, they would intersect S a second time. Hence, S_1 and S_n have also zero inner edges and we have at least three pseudosegments with zero inner edges in total.

Case 2: v_1 and v_n are incident to the same pseudosegment S having one inner edge. Let the other pseudosegments being incident to v_1 and v_n be S_1 and S_n, respectively (clearly, they are distinct); see Figure 4.3c. Since S has an inner edge, S_1 and S_n have zero inner edges. Consider the face f following the inner edge e of S. Beside S, the two other distinct bounding pseudosegments of f are S_2 and S_3. Let S_3 have an inner edge e_3 following e along the sequence of inner faces. All faces of the outerpath are incident to S, hence S_3 cannot have a second inner edge as it intersects S incident to f. Similarly, S_2 can have at most one inner edge e_2 when it intersects S incident to f. Thus, S_1 and S_n have zero inner edges, while S, S_2, and S_3 have at most one inner edge each.



(a) Case 0. (cannot occur) **(b)** Case 1. **(c)** Case 2. **(d)** Case 3. **(e)** Case 4. **(f)** Case 5. **(g)** Case 6.

Figure 4.3: Cases to show Lemma 4.5. The blue arrow indicates the path of faces.

Case 3: v_1 and v_n are incident to the same pseudosegment S having at least two inner edges. As in Case 2, when the sequence of faces of the outerpath passes over S, there are two pseudosegments S_2 and S_3 each having at most one inner edge. We have this situation at least twice – we denote the next corresponding pair of segments that has at most one inner edge per pseudosegment by S_4 and S_5. Observe that maybe $S_3 = S_4$; see Figure 4.3d. Then, however, $S_2 \neq S_5$ as otherwise S_2 and S_3 would intersect twice. So we have two pseudosegments with zero inner edges (S_1 and S_n) and we have ≥ 3 pseudosegments with at most one inner edge (S_2, S_3, and S_5).

Case 4: v_1 and v_n are incident to four distinct pseudosegments and exactly one of these pseudosegments has at least two inner edges. This case is similar to Case 2 and Case 3. Without loss of generality, let the segments of v_1 and v_n be $S_{1,l}$, $S_{1,r}$ and $S_{n,l}$, $S_{n,r}$, respectively, and let $S_{1,l}$ have at least two inner edges; see Figure 4.3e. Consider the first inner edge e of $S_{1,l}$ and the face f preceding e along the sequence of faces. Let the other pseudosegments bounding f be S_1 and S_2 and let the inner edge e_1 for entering f be contained in S_1. Until the sequence of faces passes over $S_{1,l}$ a second time, all faces are neighboring $S_{1,l}$. Hence, S_1 and S_2 have at most one inner edge each. Moreover, observe that neither $S_{n,l}$ nor $S_{n,r}$ can be equal to S_1 or S_2 as otherwise they would intersect $S_{1,l}$ twice. This gives us our bound – the three pseudosegments with at most one inner edge are S_1, S_2, and one of $S_{n,l}$ and $S_{n,r}$.

Case 5: v_1 and v_n are incident to four distinct pseudosegments and two of these pseudosegments have at least two inner edges. We have a very similar situation as in Case 4, but now we have S_1 and S_2 in the forward direction and S'_1 and S'_2 symmetrically in the backward direction; see Figure 4.3f. Let $S_{1,l}$ and $S_{n,l}$ be the segments originating at v_1 and v_n, respectively, that have at least two inner edges each. We have to be a bit more careful about the case that $S_{1,l}$ and $S_{n,l}$ intersect. However, even in this case S_1, S_2, S'_1, and S'_2 are four distinct pseudosegments since the first inner edge e of $S_{1,l}$ precedes all inner edges of $S_{n,l}$ and the last inner edge e' of $S_{n,l}$ succeeds all inner edges of $S_{1,l}$.

Case 6: v_1 and v_n are incident to four distinct pseudosegment and each of them has at most one inner edge. If three of them have zero inner edges, we are done. So assume that the pseudosegment S_l of v_1 (and one pseudosegment of v_n) has an inner edge e; see Figure 4.3g. Consider the face f preceding e in the sequence of faces in the outerpath. Beside S_l, let f be bounded by S_1 and S_2. The key insight is that S_1 and S_2 pass over S_l incident to e, but on the other side of S_l, they cannot intersect a second time and so the path of faces in the outerpath can yield another inner edge at most for one of S_1 and S_2. Hence, either both S_1 and S_2 have at most one inner edge or S_2 has zero inner edges, which provides our bound. We have to be careful about the case that S_1 or S_2 are pseudosegments of v_n. Note that not both of them can reach v_n because then they would intersect each other a second time. If S_2 reaches v_n, then S_1 is our third pseudosegment with at most one inner edge. If S_1 reaches v_n, then S_2 is our third pseudosegment without inner edges. □

We use Lemma 4.5 to fill the gaps in Equation (4.9) for $k = 1$, from which we then obtain Theorem 4.6.

Theorem 4.6. *For any n-vertex maximal outerpath G, it holds that* $\operatorname{seg}(G) \geq \operatorname{arc}_1(G) \geq \lceil \frac{n}{2} \rceil + 2$.

Proof. Since $\operatorname{seg}(G) \geq \operatorname{arc}_1(G)$, it suffices to show that $\operatorname{arc}_1(G) \geq \lceil \frac{n}{2} \rceil + 2$. Using Lemma 4.5, we observe that $3\operatorname{arc}_1^0 + \operatorname{arc}_1^1 \geq 9$. If we plug this into Equation (4.9) and set $k = 1$, we obtain

$$\operatorname{arc}_1 \geq \frac{2n - 2 + 3\operatorname{arc}_1^0 + \operatorname{arc}_1^1}{4} = \frac{n}{2} + \frac{7}{4} .$$

As we cannot have partial (pseudo)segments, we can round up this term, which gives, for all natural numbers n, $\operatorname{arc}_1 \geq \lceil \frac{n}{2} \rceil + 2$. □

Arcs and Pseudo-Circular Arcs. With $k = 2$, we continue with the next version of pseudo-k-arc arrangements, which have as a natural counterpart circles and circular arcs. The general formula in Equation (4.9) leads to the following lower bound. We have no finer analysis for the values of arc_2^i, so we assume that they can be 0, which may give us a slightly weaker bound than for (pseudo)segments.

Theorem 4.7. *For any n-vertex maximal outerpath G, it holds that* $\operatorname{arc}(G) \geq \operatorname{arc}_2(G) \geq \lceil \frac{2n+1}{7} \rceil$.

Proof. Since $\operatorname{arc}(G) \geq \operatorname{arc}_2(G)$, it suffices to show that $\operatorname{arc}_2(G) \geq \lceil \frac{2n+1}{7} \rceil$.
Using $k = 2$ in Equation (4.9) yields

$$\operatorname{arc}_2 \geq \frac{2n + 1 + 5 \operatorname{arc}_2^0 + 3 \operatorname{arc}_2^1 + 1 \operatorname{arc}_2^2}{7} \geq \frac{2n+1}{7} .$$

Since we can only have an integral number of arcs, we can again round up this value. □

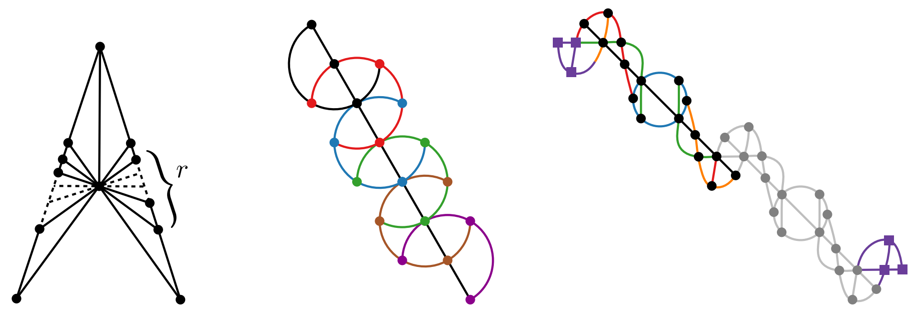
Curve Arrangements. For $k > 2$, it is not obvious how to generalize circular arcs. Still, we can make a similar statement for curve arrangements, which follows directly from Equation (4.9).

Proposition 4.8. *Let G be an n-vertex maximal outerpath drawn on a curve arrangement in the plane such that (i) any two curves intersect at most k times, and (ii) curves can be closed but do not self-intersect. Then,* $\operatorname{arc}_k(G) \geq \lceil \frac{2n-6}{3k+1} \rceil + 1$.

Tightness. Now that we have established universal lower bounds, it is a natural question to ask whether these bounds are tight. To this end, we consider existential upper bounds for segments and arcs. The infinite families of examples in Proposition 4.9 and Figure 4.4a show that our bound for segments is tight. This implies, somewhat surprisingly, that, at least for worst-case instances, using pseudosegments requires as many elements as using straight-line segments. Whether this also holds for pseudo-circular arcs and circular arcs is an open question. With circular arcs, we could not beat a bound of $n/3$, which we could do for pseudo-circular arcs. However, for both we did not reach $2n/7 \approx 0.2857n$, which might be seen as a weak indication that the upper bound in Theorem 4.7 is not tight.

Proposition 4.9. *For every $r \in \mathbb{N}^+$, there exist maximal outerpaths P_r, Q_r, U_r s.t.*
(i) P_r has $n = 2r + 6$ vertices and $\operatorname{seg}(P_r) \leq r + 5 = \frac{n}{2} + 2$,
(ii) Q_r has $n = 3r$ vertices and $\operatorname{arc}(Q_r) \leq r + 1 = \frac{n}{3} + 1$,
(iii) U_r has $n = 16r + 6$ vertices and $\operatorname{arc}_2(U_r) \leq 5r + 3 = \frac{5n}{16} + \frac{9}{8} \approx 0.3125\, n$.

Proof. We consider each of these families of maximal outerpaths individually.
(i) Consider Figure 4.4a. The maximal outerpath P_0 consists of five segments and six vertices: At the first and the last vertex, there originate on both sides two segments, which we call *long* and *short leg*. The four pairwise intersection points of these legs determine the other four vertices. At the intersection point of the short legs, there is a central vertex having an edge (the fifth segment) to the intersection point of the two long legs. Now for $r > 0$, we have r additional segments going through the central vertex and connecting the long legs. Each of them contributes one segment and two vertices. Hence the number of vertices is $n = 6 + 2r$ and the number of segments is $5 + r = n/2 + 2$.



(a) P_r: $n/2+2$ segments (matching the lower bound in Theorem 4.6)

(b) Q_6: $n/3+1$ circular arcs

(c) U_2: $5n/16+9/8 < n/3$ pseudo-circular arcs

Figure 4.4: Infinite families of maximal outerpaths requiring only a small number of segments, circular arcs, or pseudo-circular arcs in relation to the number of vertices.

(ii) Consider Figure 4.4b, where $r = 6$. The maximal outerpath Q_r has as a *base segment* (i.e., a circular arc with infinite radius), onto which r circles are aligned such that each circle has two intersection points with the previous circle, two intersection points with the next circle, and two intersection points with the base segments, which are at the same time touching points with the second circle before (after). (Of course, the first (last) circle does not have intersection points with a predecessor (successor).) These intersection points define our set of vertices. So, we have $n = 3r$ vertices and $r + 1 = n/3 + 1$ segments.

(iii) Consider Figure 4.4c. The *base structure* is colored there and the two endings are purple squares. Together they form the maximal outerpath U_1, which is similar to Q_r in that it has a base segment in the middle, but the pseudo-circular arcs on top of it appear less regular. If we increase r by one, we add a base structure between the last added base structure and one of the endings. Note that U_1 has 22 vertices and 8 pseudo-circular arcs. With each additional base structure, we gain 16 vertices and 5 pseudo-circular arcs. Hence, we have $n = 16r + 6$ and $\text{arc}_2 \leq 5r + 3 = 5n/16 + 9/8$. Also, observe that each pair of pseudo-circular arcs intersects at most twice. □

4.3 Maximal Outerplanar Graphs and 2-Trees

In the previous section, we have established a universal lower bound on the segment number for maximal outerpaths, which generalizes to circular arcs and pseudo-k-arcs. However, it is not obvious how to extend these results from maximal outerpaths to more general graph classes like maximal outerplanar graphs or 2-trees. Actually, if we prove our result regarding the universal lower bound on the segment number of maximal outerpaths by a different approach, we can also obtain a universal lower bound for maximal outerplanar graphs and for 2-trees. This different approach is counting ports at the vertices, which we can then use to “glue” outerpaths in a clever

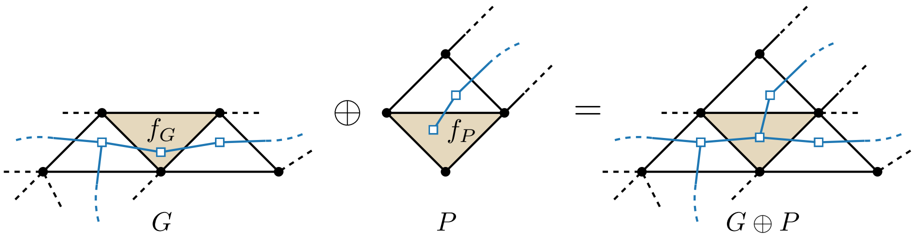


Figure 4.5: Gluing drawings of an outerplanar graph G and an outerpath P.

way to construct maximal outerplanar graphs and 2-trees. However, for this approach, which we present in this section, it is not obvious how to extend it to circular arcs (or pseudo-k-arcs).

Universal Lower Bound. Consider a straight-line drawing Γ_G of a 2-tree G. The main idea for a universal lower bound for 2-trees (and for its subclass of maximal outerplanar graphs) is that G either has many degree-2 vertices and thus requires many segments (recall that, in a 2-tree, all faces are triangles, hence degree-2 vertices cannot be closed) or G can be obtained by gluing few outerpaths for which we know (tight) universal lower bounds on the segment number. By gluing we mean the following. Let G be a 2-tree and P a maximal outerpath. Let f_G be a triangle of G that is not incident to a degree-2 vertex and let f_P be a triangle of P that is incident to a degree-2 vertex (i.e., f_P is the first or last triangle of P). Let Γ_P be a straight-line drawing of P. Then we define the *gluing* of Γ_P to Γ_G as the straight-line drawing $\Gamma_{G \oplus P}$ of the 2-tree $G \oplus P$ obtained by identifying f_P and f_G; see Figure 4.5. Note that $|V(G \oplus P)| = |V(G)| + |V(P)| - 3$. In $\Gamma_{G\oplus P}$, we call f_G and f_P the *gluing faces* of G and P, respectively.

Unfortunately, for gluing outerpaths, we cannot directly employ Theorem 4.6 because it does not tell us how many ports we lose when gluing. So, we first investigate the distribution of ports within a straight-line drawing of a maximal outerpath. We will see that, by some careful counting arguments, we lose only few (counted) ports when gluing outerpaths. We start by formally proving some auxiliary properties.

Lemma 4.10. *Let P be a maximal outerpath given together with a stacking order $\langle v_1, v_2, \dots, v_n \rangle$, and let v be a vertex of P. Then, in any outerplanar straight-line drawing of P, all of the following holds.*

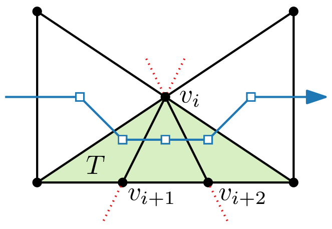
(P1) *If* $\deg(v) = 2$ *or* $\deg(v)$ *is odd, then v is open.*

(P2) *If* $\deg(v) \geq 5$*, then v is succeeded by* $\deg(v) - 4$ *many neighbors of degree* 3*, which we call* companions.

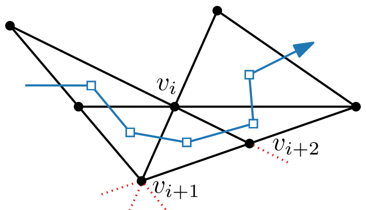
(P3) *If* $\deg(v) \geq 6$ *and v is closed, then v has a companion with three ports, which we call* bend companion.

(P4) *If subsequent vertices u and v both have degree 4, then u or v is open.*

(P5) *Let v be stacked upon the edge uw and u, v be subsequent vertices. If v is closed,* $\deg(v) = 4$*,* $\deg(u) = 3$*, and* $\deg(w) = 5$*, then either u or w has at least three ports.*

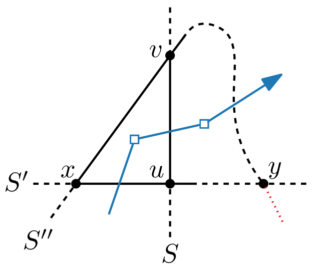


(a) If all companions of v_i are collinear, then v_i cannot be closed.

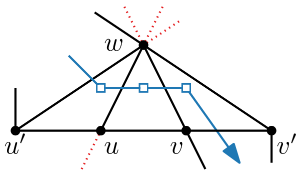


(b) If v_i is closed, then it has a bend companion (in this case v_{i+1}).

Figure 4.6: For (P3) in Lemma 4.10, we consider a vertex v_i with degree at least 6 and even.



(a) For (P4), two subsequent vertices u and v of degree 4 cannot both be closed because of the two triangles with their common neighbors x and y.



(b) For (P5), in case u has only one port, w has at least three ports because four of its neighbors are collinear.

Figure 4.7: Configurations in the proof of Lemma 4.10 where v is closed and has degree 4.

Proof. We consider each of the statements individually. This suffices to prove the correctness of the lemma.

(P1) If $\deg(v)$ is odd, the claim is trivially true. Otherwise, v and its two neighbors form a triangle in any 2-tree and cannot be collinear.

(P2) Since v has degree at least five, constructing P with a sequence of stacking operations involves $\deg(v) - 2$ consecutive stacking operation on edges incident to v. Consequently, all succeeding neighbors of v, except for the last two, must have degree three.

(P3) Let $v_i = v$. Consider the companions $v_{i+1}, \ldots, v_{i+\deg(v_i)-4}$ of v. Suppose neither of them has three ports (two is not possible since they have degree 3; see (P2)), then (at least) $\deg(v_i) - 2$ neighbors of v_i are collinear and thus result in a triangle T with v at one corner and these neighbors on the opposing side of T; see Figure 4.6a. Then, however, v_i cannot be closed since $\deg(v_i) - 2 > \deg(v_i)/2$ and at most two segments can pass through v_i, which is a contradiction. Hence, one of the companion vertices of v has three ports; see Figure 4.6b.

(P4) Let $\langle x, u, v, y \rangle$ be a stacking subsequence in P where both u and v have degree 4; see Figure 4.7. Assume, for the sake of contradiction, that there exists a planar straight-line drawing of P where both u and v are closed. Let S be the segment that contains the edge uv. Then S intersects at u the segment S' that contains xu, and S intersects at v the segment S'' that contains xv; see Figure 4.7a. Observe that S' and S'' need to intersect again in y since P is a maximal outerpath and both u and v are closed. However, this would only be possible if x, u, v, and y are collinear, which is a contradiction to the drawing being a planar straight-line drawing.

(P5) If u has three ports, we are done. Otherwise, u has only one port; see Figure 4.7b. Then note that u and v need to be collinear with a successor v' of v and predecessor u' of u. Observe that these four vertices are adjacent to w. However, since w has degree 5, only one of the edges $\{u', w\}, \{u, w\}, \{v, w\}, \{v', w\}$ can be extended at w. (In Figure 4.7b, the edge $\{v', w\}$ lies on a segment passing through w.) Therefore, w has at least three ports. □

Proposition 4.11. *Let P be a maximal outerpath with $n \geq 4$ vertices. Then* $\text{port}(P) \geq n + 1$. *Moreover, for any planar straight-line drawing of P, we can find a bijective (but not necessarily total) assignment of ports to vertices such that every port is assigned to its own vertex or to a neighboring vertex.*

Proof. Given any straight-line drawing Γ_P and any stacking order $\langle v_1, \ldots, v_n \rangle$ of P, we describe an assignment of ports to vertices in their vicinities such that no two ports are assigned to the same vertex and we have a port assigned to every vertex. This immediately proves that $\text{port}(P) \geq n$. For the one remaining port, we take an unassigned port at v_1.

Let $i \in \{1, \ldots, n\}$. We consider all different situations for vertex v_i. Each situation is illustrated by a vertex in the example shown in Figure 4.8 (to which we refer in brackets). If v_i is open (such as v_1, v_2, or v_4), we assign one of the ports to itself. If v_i is closed, then $\deg(v_i)$ is even and at least 4 by (P1).

First assume $\deg(v_i) \geq 6$ (such as v_9, v_{13}, v_{16}). Then, by (P3), we know that v_i has a bend companion v_j with three ports. Only one of the three ports of v_j is assigned to v_j itself, so we assign one of the remaining ports of v_j to v_i (such a port is supplied by v_{11}, v_{14} and v_{18}).

If $\deg(v_i) = 4$, then either $\deg(v_{i-1}) = 4$ (such as v_4 preceding v_5) or $\deg(v_{i-1}) = 3$ (such as v_{11} preceding v_{12} and v_2 preceeding v_3) since, by (P2), v_{i-1} has degree at most 4. In the former case, v_{i-1} has at least two ports by (P4) and we can assign one of the ports to v_i (such as v_4 to v_5). In the latter case, we distinguish three subcases. If $v_i = v_3$ in the stacking order of P, then v_2 has degree 3 and cannot be closed (as v_2 and v_3 in Figure 4.8). If $v_i = v_4$ in the stacking order of P, then v_2 or v_3 has three ports. Otherwise, observe that the common neighboring predecessor of v_{i-1} and v_i has degree at least 5; hence one of (P3) or (P5) applies (see v_9, v_{11}, and v_{12}). This is the only case where a vertex may provide ports for itself and two other vertices. □

Proposition 4.11 also implies a universal lower bound of $(n+1)/2$ for the segment number of n-vertex maximal outerpaths. In Theorem 4.6, we have improved this by a constant to obtain a tight universal lower bound.

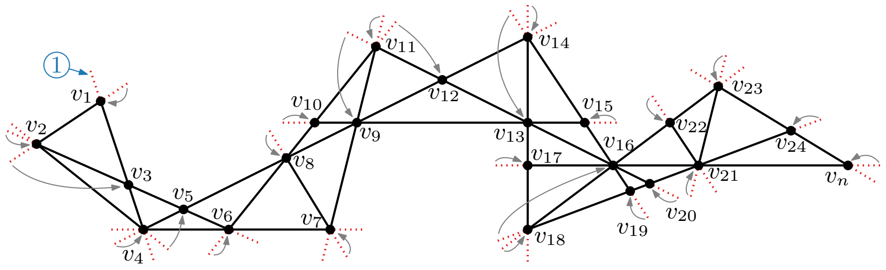


Figure 4.8: A straight-line drawing of a maximal outerpath where each vertex is assigned a port (indicated by grey arrows). Several ports remain unassigned (e.g., ①).

We can use Proposition 4.11 as a tool to extend our graph class from outertpaths to maximal outerplanar graphs. This leads us to the main theorem of this section, which we prove next.

Theorem 4.12. *For any 2-tree (or any maximal outerplanar graph) G with n vertices,* $\operatorname{seg}(G) \geq \lceil \frac{n+7}{5} \rceil$.

Proof. For now, assume that G is a maximal outerplanar graph. We consider the case that G is a 2-tree at the end of this proof. If the weak dual tree T of G has at least $(n+7)/5$ leaves, we are done since G has at least as many segments as T has leaves.

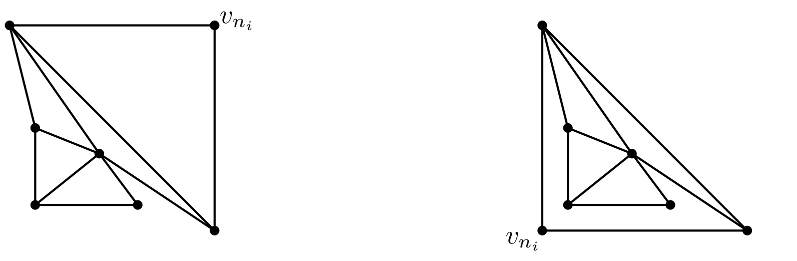
Otherwise, let $\mathcal{P} = \{P_1, \dots, P_\ell\}$ be a minimum-size set of maximal outerpaths such that when we define $G_1 = P_1$ and $G_i = G_{i-1} \oplus P_i$, for $i \in \{2, \dots \ell\}$, we get that $G = G_\ell$. In other words, we can obtain G by $\ell - 1$ consecutive gluing operations of the paths in $\mathcal{P}$. Note that $\ell \leq (n+7)/5 - 1 = (n+2)/5$ because P_1 contains two leaves and, for $i \in \{2, \dots \ell\}$, P_i contains one leaf of T.

Next, we show a lower bound on the number of ports on any straight-line drawing of G. To this end, we use the assignment of ports to vertices that we established in Proposition 4.11 and apply it to each outerpath P_i in $\mathcal{P}$. Further, we use the stacking order of P_i that starts at the degree-2 vertex of P_i that is not incident to the gluing face of P_i. For $i \in \{1, \dots, \ell\}$, let n_i be the number of vertices in P_i. Note that for G, the number of vertices is $n = \sum_{i=1}^{\ell} n_i - 3(\ell - 1)$.

First, we compute $\operatorname{port}(\mathcal{P})$, the sum of ports counted for $P_1, \dots, P_\ell$:

$$\operatorname{port}(\mathcal{P}) = \sum_{i=1}^{\ell} \operatorname{port}(P_i) \geq \sum_{i=1}^{\ell} (n_i + 1) = n + 3(\ell - 1) + \ell = n + 4\ell - 3$$

Second, we analyze the number of *counted* ports that we lose by the $\ell - 1$ gluing operations. Consider the gluing operation $G_i = G_{i-1} \oplus P_i$ and let $f_{G_{i-1}}$ and f_P be the gluing faces of G_{i-1} and P, respectively, identified to face f of G_i. Observe that we counted three ports at f_P since neither v_{n_i} nor one of its neighbors needs to assign a port to another vertex (we assign only ports to vertices coming later in the stacking order except for bend companions, but the last three vertices cannot be bend



(a) Initial outerplanar drawing. **(b)** Drawing after flipping v_{n_i}.

Figure 4.9: Straight-line drawing of a maximal outerpath where we “flip” v_{n_i} over the rest of the drawing such that the resulting drawing remains planar. This way, we can append maximal outerpaths to inner faces of 2-trees.

companions). We assume to lose all of these three ports when gluing. This means that every vertex has at most as many counted ports in G_i as it had in G_{i-1}. For the ports lost at $f_{G_{i-1}}$, observe that the vertex that is identified with v_{n_i} at P_i cannot lose any ports. The other two vertices are neighbors in G_{i-1}. In the assignment from Proposition 4.11, any two such vertices provide ports for at most four vertices in total. We assume also to lose all of these ports, which results in a total loss of at most seven ports per gluing operation. Hence, with $\ell \leq (n+2)/5$, we get

$$\operatorname{seg}(G) = \frac{\operatorname{port}(G)}{2} \geq \frac{\operatorname{port}(\mathcal{P}) - \text{loss}}{2} \geq \frac{n + 4\ell - 3 - (7\ell - 7)}{2} \geq \frac{n+7}{5}\,.$$

It remains to consider the case that G is a 2-tree. As for maximal outerplanar graphs, we can also construct a 2-tree by gluing multiple outerpaths. Similar to leaves in the weak dual tree, each attached outerpath provides at its ending a vertex of degree 2 with two ports. The only exception is that we are not restricted on gluing to the outside – we may also draw an outerpath within an inner face of the current 2-tree drawing. A difficulty is how to identify the faces $f_{G_{i-1}}$ and f_P if we want to draw the rest of P within this unified face. However, consider an outerplanar straight-line drawing of P where we “flip” the last vertex v_{n_i} over the rest of the drawing such that the drawing remains planar; see Figure 4.9. Clearly, the number of ports in the drawing of the maximal outerpath P did not change and the assignment scheme from Proposition 4.11 is still applicable. We may use such flips also along inner edges of an outerpath drawing to obtain a “folded” outerpath drawing with the same properties. Hence, we can apply gluing operations to inner faces with at most the same loss as analyzed before. □

Tightness. We remark that, though we get the same lower bound for maximal outerplanar graphs and 2-trees, the actual (tight) numbers might be different. In other words, maybe there are 2-trees requiring less segments than any maximal outerplanar graph with the same number of vertices. Our current analysis is already for maximal outerplanar graphs most likely not tight as we see by comparison with our existential upper bound for the universal lower bound in Figure 4.10. The

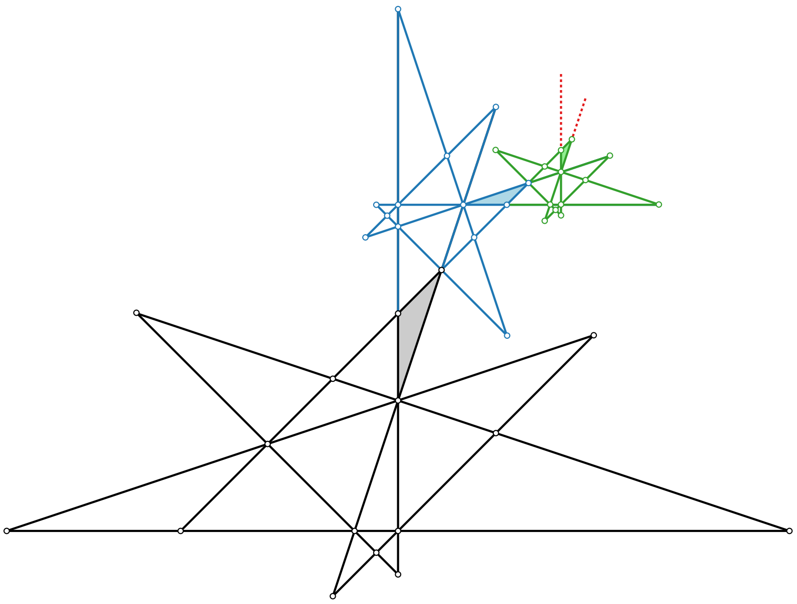


Figure 4.10: The maximal outerplanar graph O_3 with 42 vertices drawn on 18 segments.

construction there defines a family of graphs $O_1, O_2, \ldots$ where the base graph O_1 has 16 vertices and admits a drawing Γ_{O_1} with 8 segments. From O_{i-1} to O_i, we glue a scaled and rotated copy of Γ_{O_1} to the drawing of O_{i-1} (gluing faces are shaded). In each step, we get 13 more vertices with only 5 more segments and hence the following result.

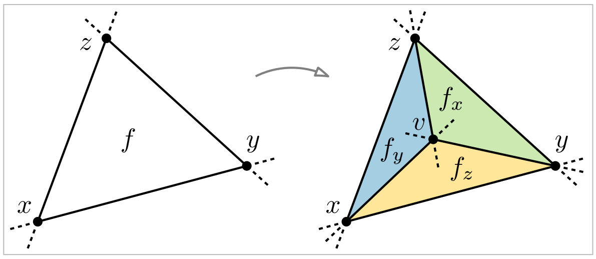
Proposition 4.13. *For every $r \in \mathbb{N}^+$, there exists a maximal outerplanar graph O_r that has $n = 13r + 3$ vertices and* $\operatorname{seg}(O_r) \leq 5r + 3 = \frac{5n+24}{13}$.

4.4 Planar 3-Trees

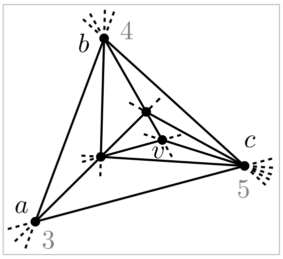
Previously, we have studied the segment number of 2-trees and we have found universal and existential bounds. This rises the question whether we can obtain similar bounds for planar 3-trees.

Universal Lower Bound. Finding a universal lower bound for planar 3-trees seems to be easier than for 2-trees: for a planar 3-tree G with $n \geq 6$ vertices and an arbitrary planar straight-line drawing Γ of G, we observe that we can assign at least (i) one port to each inner face of Γ and (ii) twelve ports to the outer face of Γ; see Figure 4.11. By Euler, any n-vertex triangulation has $2n - 5$ inner faces. Thus, Γ has $2n + 7$ ports. This yields the following bound, which is tight up to a constant.

Theorem 4.14. *For any planar 3-tree G with $n \geq 6$ vertices,* $\operatorname{seg}(G) \geq n + 4$.



(a) Stacking a vertex v into an inner face $f = \langle x, y, z \rangle$ creates a port in each of the new faces f_x, f_y, and f_z.



(b) A planar 3-tree with $n \geq 6$ vertices has at least 12 ports on the outer face $\langle a, b, c \rangle$.

Figure 4.11: For planar 3-trees we can establish a simple universal lower bound on the segment number by counting ports in inner faces and ports on the outer face.

Proof. For (i), consider a sequence of stacking operations that starts with a drawing of K_4 and yields Γ. Let v be the current vertex in this process, and let f be the face into which v is stacked. Let $V(f) = \{x, y, z\}$ be the set of vertices incident to f, and let f_x, f_y, and f_z be the three newly created faces such that $V(f_x) = \{v, y, z\}$ etc.; see Figure 4.11a. Since f is a triangle, no two of the edges xv, yv, zv can share a segment. Thus, v has three ports. In particular, the segment of xv points into f_x, the segment of yv points into f_y, and the segment of zv points into f_z. We assign the ports of v accordingly to f_x, f_y, and f_z. When the stacking process ends with Γ, each inner face of Γ has a port assigned to it.

For (ii), note that the number of ports on the outer face equals the sum of the degrees of the three vertices on the outer face. K_4 has nine ports. The fifth vertex in the stacking order is incident to two vertices on the outer face, so it contributes two ports. Similarly, the sixth vertex contributes at least one port; see Figure 4.11b. Thus, in total, the outer face has at least twelve ports. (This bound is tight since any further vertex can be stacked into a face that is not adjacent to the outer face.) ☐

Tightness. After determining this universal lower bound on the segment number of 3-trees, we next show that this bound is tight up to an additive constant of 3. To this end, we construct an infinite family of planar 3-trees using $n + 7$ segments where n is the number of vertices; see Figure 4.12. Next, we formalize this construction.

Proposition 4.15. *For every $r \in \mathbb{N}^+$, there exists a planar 3-tree T_r that has $n = 4r + 8$ vertices and* $\operatorname{seg}(T_r) \leq 4r + 15 = n + 7$.

Proof. Consider Figure 4.12. We start by drawing the outer triangle $\langle v_1, v_2, v_3 \rangle$ using three segments. As fourth vertex, we add the central vertex x introducing three more segments. For the fifth and sixth vertex, u and w, we re-use the line segments xv_1 and xv_2 and, consequently, add only four new segments. For the seventh and eighth vertex, y and z, we re-use line segments uv_3 and wv_3, respectively. Moreover, they share a segment for the edges yx and zx, which results in three new segments. This gives us 8 vertices 13 segments for the base construction.

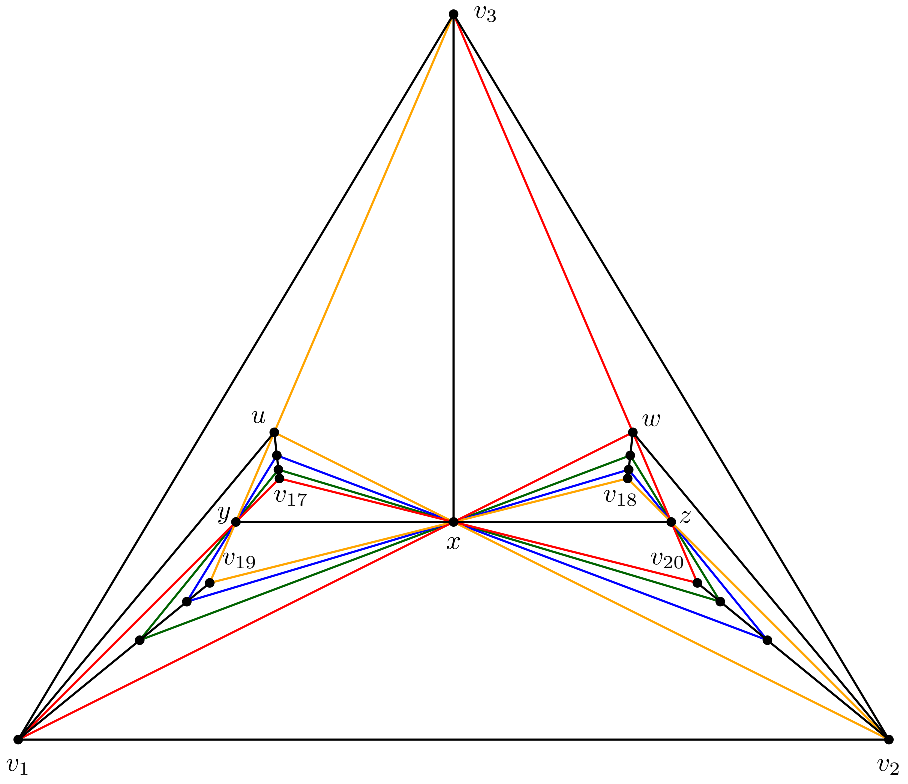


Figure 4.12: The planar 3-tree T_3 with 20 vertices drawn on 27 segments.

Now in r rounds, we iteratively stack four vertices into the faces $\langle u, y, x\rangle$, $\langle w, x, z\rangle$, $\langle v_1, x, y\rangle$, and $\langle v_2, z, x\rangle$. We stack along four new (black) line segments (see e.g., $\overline{v_1 v_{19}}$ in Figure 4.12) such that the final drawing uses four more segments once as well as four more per iteration (one per two edges; see the colored line segments through y, z and x in Figure 4.12). We re-use the segments of uy, wz, $v_1 y$, and $v_2 z$ for one edge each, which saves us two more segments. Together with the 13 segments of the base construction, we get $\operatorname{seg}(T_k) \leq 13 + 4 + 4k - 2 = 4k + 15 = n + 7$. $\square$

Existential Lower Bound on the Universal Upper Bound. Consider the universal upper bound of $2n - 2$ on the segment number of planar 3-trees due to Dujmović et al. [DESW07, Lemma 18]. They show the tightness of their result in a fixed-embedding scenario, that is, they prove that there is a family $(B_n)_{n\geq 4}$ of *plane* 3-trees (see Figure 4.13a) such that B_n has n vertices and requires $2n - 2$ segments in any straight-line drawing that adheres to the given embedding. They remark that, given a different embedding, B_n can be drawn using roughly $3n/2$ segments; see Figure 4.13c. We formalize this to compute the exact segment number of B_n, which will be useful in Section 4.5.

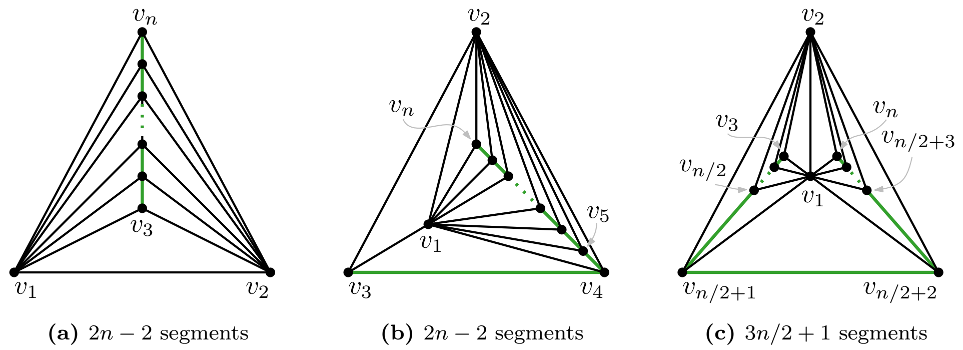


Figure 4.13: Straight-line drawings of the 3-tree B_n (with $n \geq 6$ and even) for two different embeddings that were analyzed by Dujmović et al. [DESW07]

Proposition 4.16. *For every $n \in \mathbb{N}$ where $n \geq 6$, there exists a 3-tree B_n that has n vertices and* $\operatorname{seg}(B_n) = \lceil \frac{3n}{2} \rceil + 1$.

Proof. We first show the lower bound $\operatorname{seg}(B_n) \geq \lceil 3n/2 \rceil + 1$. Let B_n be the graph depicted in Figure 4.13 with vertex set $\{v_1, v_2, \ldots, v_n\}$ and edge set $\{v_1v_i, v_2v_i \colon 3 \leq i \leq n\} \cup \{v_iv_{i+1} \colon 1 \leq i \leq n-1\}$. If v_1 and v_2 are on the outer face (see Figure 4.13a), we have at least $2(n-2)+2$ segments, but for $n \geq 6$, $2n-2 \geq \lceil 3n/2 \rceil + 1$.

So w.l.o.g. let v_1 not be on the outer face. Consequently, v_2 lies on the outer face because any triangle of the graph contains at least one vertex of $\{v_1, v_2\}$ and, hence, also the triangle of the outer face. This implies that there are $n-1$ distinct segments incident to v_2. For every $i \in \{3, \ldots, n\}$, the path $\langle v_1, v_i, v_2 \rangle$ is drawn with a bend at v_i because otherwise, it would coincide with the edge v_1v_2. Therefore, the $n-2$ edges $v_1v_3, \ldots, v_1v_n$ form at least $(n-2)/2$ new segments.

Now consider the two other vertices on the outer face – we call them u and w. The edge uw yields another segment. Moreover, v_3 and v_n cannot both be on the outer face as they are not adjacent. Therefore, w.l.o.g., u has degree 4. So far, we have counted the segments of the edges uv_1, uv_2 and uw. This means that there is another segment for the fourth edge incident to u.

If w has degree 4, as well, we count another segment by the same argument. Overall, this sums up to at least $(n-1)+(n-2)/2+1+2 = 3n/2+1$ segments. Otherwise, w has degree 3. Assume w.l.o.g. that $w = v_3$. Consequently, the outer face is the triangle $\langle v_2, v_4, v_3 \rangle$; see Figure 4.13b. Observe now that $\langle v_1, v_2, v_4 \rangle$ separates v_3 on the outside from all other vertices in the inside. Thus, the $n-3$ edges $v_1v_4, \ldots, v_1v_n$ reach v_1 in an angle smaller than 180° and, hence, require $n-3$ distinct segments. This results in at least $(n-1)+(n-3)+1+1 = 2n-2 \geq 3n/2+1$ segments.

Finally, we show that this lower bound is tight. Consider the drawing of B_n in Figure 4.13c. It uses exactly the $\lceil 3n/2 \rceil + 1$ segments that we counted above for the lower bound. In particular, $u = v_{\lfloor n/2 \rfloor + 1}$ and $w = v_{\lfloor n/2 \rfloor + 2}$. □

4.5 The Ratio of Segment Number and Arc Number

Since circular-arc drawings are a natural generalization of straight-line drawings, it is natural to also ask about the maximum ratio between the segment number and the arc number of a graph. In other words, *how much can we save if we use circular arcs instead of line segments*?

In this section, we make some initial observations regarding this question. Clearly, for any graph G, $\operatorname{seg}(G)/\operatorname{arc}(G) \geq 1$. Note that $\operatorname{seg}(K_3)/\operatorname{arc}(K_3) = 3$ since we can place three vertices on a single circle, but we need three line segments. We investigate the ratio for two classes of planar graphs, for which we construct families of graphs to show the following.

- For maximal outerpaths, (and, hence, for maximal outerplanar graphs and 2-trees) the minimum ratio is 1 (Proposition 4.17/Figure 4.4a), i.e., there are maximal outerpaths where using circular arcs instead of line segments saves nothing (except for maybe a small additive constant), and
- the maximum ratio is at least 2 (Proposition 4.18/Figure 4.4b), i.e., there are maximal outerpaths where we need at most half as many arcs as segments.
- For planar 3-trees, the minimum ratio is at most 4/3 (Proposition 4.19/Figure 4.12), and
- the maximum ratio is at least 3 (Proposition 4.20/Figure 4.14).

It is open how much of an improvement in terms of visual complexity circular-arc drawings offer over straight-line drawings for arbitrary planar graphs. Can the ratio between segment and arc number be bounded by 3 for every planar graph?

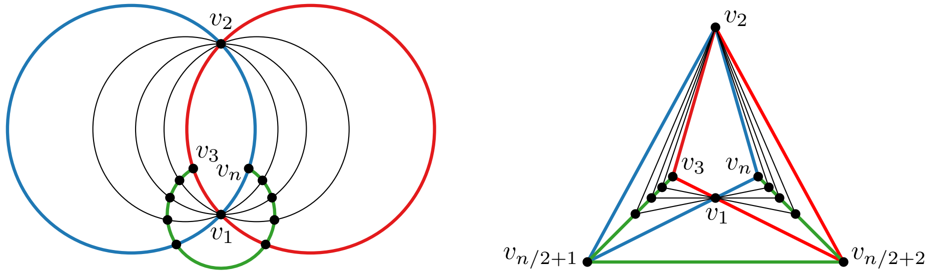
Proposition 4.17. *For $r \in \mathbb{N}^+$, let P_r be the maximal outerpath from Proposition 4.9 and Figure 4.4a. Then,* $\lim_{r\to\infty} \frac{\operatorname{seg}(P_r)}{\operatorname{arc}(P_r)} = 1$.

Proof. Consider Figure 4.4a for a drawing of P_r on $n/2+2$ segments where n is even and is the number of vertices of P_r. Observe that the central vertex has degree $(n-1)$ and, thus, is contained in at least $n/2$ different arcs in any arc-drawing. Hence, the segment number and the arc number of P_n differ by at most a constant of 2. $\square$

Proposition 4.18. *For $r \in \mathbb{N}^+$, let Q_r be the maximal outerpath from Proposition 4.9 and Figure 4.4b. Then,* $\lim_{r\to\infty} \frac{\operatorname{seg}(Q_r)}{\operatorname{arc}(Q_r)} \geq 2$.

Proof. Consider a segment drawing of the n-vertex maximal outerpath Q_r and recall the properties of Lemma 4.10. Q_r contains $n/3-2$ degree-6 vertices and for each of them two degree-3 companions with at least one port each. The degree-6 vertices either have two ports themselves or their bend companions have three ports. In either case, we find four ports for each degree-6 vertex. The remaining six vertices around the first and the last face have at least six ports. Therefore, $\operatorname{seg}(Q_r) \geq 2n/3 - 1$. By Proposition 4.9, $\operatorname{arc}(Q_r) \leq n/3+1$ and, hence, $\operatorname{seg}(Q_r)/\operatorname{arc}(Q_r) \geq 2 - 3/(r+1)$. $\square$

Proposition 4.19. *For $r \in \mathbb{N}^+$, let T_r be the maximal outerpath from Proposition 4.15 and Figure 4.12. Then,* $\lim_{r\to\infty} \frac{\operatorname{seg}(T_r)}{\operatorname{arc}(T_r)} \leq \frac{4}{3}$.



(a) Drawing with $n/2$ (here, 6) circular arcs. **(b)** Drawing with $3n/2$ (here, 18) line segments.

Figure 4.14: The planar graph B'_n obtained from B_n in Figure 4.13 by removing the edge v_1v_2.

Proof. There is a set of $4r + 2$ unique paths, one half from y to x and the other half from z to x. Each of these paths needs to be covered by at least one arc. As they all go through x, no arc may cover more than two paths.

Now observe that any arc covering one path on each side connects all three vertices, such that only one such arc may exist (in contrast to pseudo-2-arcs). Hence, of the remaining $4r$ paths we may cover only two with the same arc (with a full circle) if both lie on the same side of x. But if we use multiple full circles that all either go through x and y or go through x and z, they all would need the same tangent in x in order not to intersect. However, these pairs of circles are unique and we can use multiple circles only on one side of x. Thus, r arcs may suffice for one side, but the other side needs $2r$ arcs, which yields a total of $3r + 1$ necessary arcs. By Proposition 4.15, $\operatorname{seg}(T_r) \leq 4r + 15$ and, hence, $\operatorname{seg}(T_r)/\operatorname{arc}(T_r) \leq 4/3 + 41/(9r + 3)$. □

Proposition 4.20. *For $n \in \mathbb{N}$ where $n \geq 8$ is even, let B'_n be the n-vertex planar graph obtained from B_n by removing the edge v_1v_2, where B_n is the planar 3-tree from Proposition 4.16 and Figure 4.13. Then,* $\frac{\operatorname{seg}(B'_n)}{\operatorname{arc}(B'_n)} = 3$ *and* $\lim_{n\to\infty} \frac{\operatorname{seg}(B_n)}{\operatorname{arc}(B_n)} = 3$.

Proof. Figure 4.14 shows drawings of B'_n with $n/2$ arcs and with $3n/2$ segments. Clearly, $\operatorname{arc}(B'_n) = n/2$ since $\deg(v_2) = n - 2$ and there are two vertices of odd degree (v_3 and v_n), where some arc(s) must start and end. For the same reason $\operatorname{arc}(B_n) = n/2 + 1$.

Recall that removing the edge v_1v_2 from B_n yields B'_n. Observe that B'_n is still triconnected. Therefore, the set of embeddings is the same as for B_n (except that we have the face $\langle v_1, v_n, v_2, v_3\rangle$ instead of the triangular faces $\langle v_1, v_2, v_3\rangle$ and $\langle v_1, v_n, v_2\rangle$) and depends only on the choice of the outer face.

Analyzing the different embeddings of B'_n as those of B_n in the proof of Proposition 4.16, shows that $\operatorname{seg}(B'_n) = \operatorname{seg}(B_n) - 1 = 3n/2$. In particular, while we could straighten the path $\langle v_2, v_3, v_1\rangle$ in Figure 4.14, this would introduce a new bend in the path $\langle v_3, v_1, v_{n/2+2}\rangle$, and the number of segments remains $3n/2$. Hence, $\operatorname{seg}(B'_n)/\operatorname{arc}(B'_n) = 3$ and $\operatorname{seg}(B_n)/\operatorname{arc}(B_n) = 3 - 4/(n + 2)$. □

4.6 An Algorithm for Cactus Graphs

We first state a lower bound for the segment number of cactus graphs. Then, we give a recursive algorithm that produces drawings matching this bound precisely.

Lemma 4.21. *Let G be any cactus graph, let η be the number of odd-degree vertices of G, and let $\gamma = 3c_0 + 2c_1 + c_2$, where c_i is the number of simple cycles with exactly i cut vertices in G. Then, $\operatorname{seg}(G) \geq \eta/2 + \gamma$.*

Proof. If G is a tree, then $\gamma = 0$ and $\operatorname{seg}(G) = \eta/2$, as shown by Dujmović et al. [DESW07]. If G is a cycle, then all vertices have degree 2 (that is, $\eta = 0$). Moreover, $c_0 = 1$ and $c_1 = c_2 = 0$. A cycle can be drawn as a triangle (but not with less than three segments), that is, $\operatorname{seg}(G) = 3$. In both cases, our claim holds.

So assume G is neither a tree nor a cycle. Then G contains at least one cycle and each cycle has at least one cut vertex, so $c_0 = 0$. Let Γ be any straight-line drawing of G. Every odd-degree vertex of G has a port in Γ. Hence, Γ has at least η ports.

Additionally, each cycle f of G is a simple polygon in Γ. Thus, f is incident to at least three segments in Γ. If f contains exactly two cut vertices, the drawing of f must contain a bend at some vertex of f that is not a cut vertex, that is, at a degree-2 vertex. This increases the number of ports by 2. Similarly, if f contains exactly one cut vertex, the drawing of f must contain two bends at degree-2 vertices, which increases the number of ports by 4. In total, Γ has at least $\eta + 4c_1 + 2c_2$ ports or $\eta/2 + 2c_1 + c_2$ segments. Since $c_0 = 0$, we have $\operatorname{seg}(G) \geq \eta/2 + 3c_0 + 2c_1 + c_2$. □

It is not difficult, but somewhat technical to draw a given cactus such that the lower bound in the above lemma is met exactly. For an idea refer to Figure 4.15.

Theorem 4.22. *Given a cactus graph G, we can compute $\operatorname{seg}(G)$ and draw G using $\operatorname{seg}(G)$ many segments in linear time. If G is given with an outerplanar embedding, the computed drawing respects the given embedding.*

Proof. If G is a tree, we use the linear-time algorithm by Dujmović et al. [DESW07], which yields a drawing with $\eta/2$ segments, which is optimal. If G is a simple cycle, we draw G as a triangle, which again is optimal. Otherwise, $c_0 = 0$. In this case, which we treat below, we draw G with $\eta/2 + 2c_1 + c_2$ segments, which is optimal according to Lemma 4.21.

We draw G recursively using its block-cut tree. We first compute the block-cut tree of G in linear time [Tar72] and root it at a cycle block node that corresponds to a simple cycle f. We start by drawing f as a regular p-gon P, where p is the maximum of 3 and the number of cut vertices of f. Let $2r$ be the side length of P, and let α be the interior angle at each corner of P. Then $\alpha = 180° \cdot (p-2)/p$.

For each cut vertex v of f, we recursively draw the subgraph $G(v)$ of G corresponding to the subtree that hangs off v in the block-cut tree; see Figure 4.15. We draw $G(v)$ into the interior of the circle $C_{v,r}$ of radius r centered at v. (Within this circle, we use only the complement of P.) For each pair of cut vertices, the interiors of the corresponding circles are disjoint; hence, the drawing of G has no edge crossings if the drawings of the subgraphs are crossing-free. Our drawing of G will have the

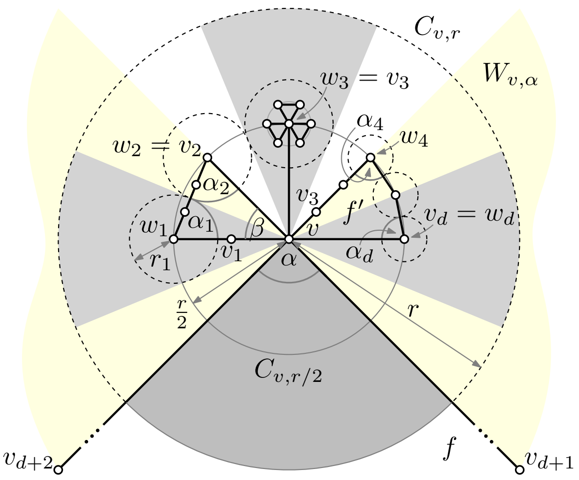


Figure 4.15: Recursive approach for drawing cactus graphs. Vertex v is a cut vertex of a cycle f (or a degree-2 vertex if f has less than three cut vertices). After f has been drawn, the algorithm recursively draws the subgraph $G(v)$ into $C_{v,r}$ such that v has a port if and only if $\deg(v)$ is odd.

following property. Each odd-degree vertex has exactly one port and, in every simple cycle of G with $j < 3$ cut vertices, there are exactly $3 - j$ degree-2 vertices with two ports. This implies that the total number of segments in our drawing meets the bound in Lemma 4.21 precisely.

Let d be the degree of v minus 2, and let v_{d+1} and v_{d+2} be the two neighbors of v that lie on f. Note that v, v_{d+1}, and v_{d+2} have already been placed. Let the neighbors $v_1, \ldots, v_{d+2}$ of v be ordered clockwise around v. We assume that neighbors that belong to the same simple cycle are consecutive in this ordering. (Note that this is the case if G is given with a fixed outerplane embedding.) We now define a set W of vertices in $G(v)$ for which we may call our algorithm recursively. Initially, W is empty. For $i \in \{1, \ldots, d\}$, if v_i and v do not lie on the same simple cycle, then set $w_i = v_i$ and add w_i to W. Otherwise let f' be the simple cycle that contains v, v_i and another neighbor of v, say, v_{i+1}. If f' does not contain a cut vertex other than v, set $w_i = v_i$ and add w_i to W. Otherwise, let w_i be the cut vertex of G closest to v_i in $G(v) - v$, i.e., $G(v)$ where we have removed v. If v_{i+1} has the same closest cut vertex w_j, then, if $w_i \neq v_i$, set $w_i = v_i$, otherwise set $w_{i+1} = v_{i+1}$. Add w_i and w_{i+1} and all other cut vertices of f' (if any) to W (except v).

We now place the vertices in W on the circle $C_{v,r/2}$. If d is odd, then we place $w_{(d+1)/2}$ on the line that bisects the angle $\angle v_{d+2} v v_{d+1}$ such that $w_{(d+1)/2}$ lies opposite of this angle (as w_3 in Figure 4.15). For the remainder of this proof, we assume for simplicity that d is even. Then $d \geq 2$ and we place $w_{d/2}$ on the line through vv_{d+1} and $w_{d/2+1}$ on the line through vv_{d+2}. We place the remaining neighbors in pairs on opposite sides of lines through v such that these lines equally partition the angle space in the double wedge $W_{v,\alpha}$ (light yellow in Figure 4.15) that is bounded by the lines through vv_{d+1} and through vv_{d+2} and does not contain the angle $\angle v_{d+2}vv_{d+1}$. The angular distance between two consecutive edges incident to v is then $\beta = (360° - 2\alpha)/d$.

We draw each simple cycle f' that contains v and two neighbors v_i and v_{i+1} of v as a simple polygon that connects v to w_i to potentially further cut vertices of f' (which we place in their order equidistant along f') to w_{i+1} to v.

Now we define, for each newly placed vertex $w \in W$ with $\deg(w) \geq 2$, values α' and r' so that we can draw the graph $G(w)$ recursively. To this end, if v and w lie on the same simple cycle f', let α' be the interior angle of w in f', and let r' be the distance of w to the closest vertex in $V(f) \cap W$ divided by 2, where $V(f)$ are the vertices of f. Otherwise, let α' be 0 and set r' such that $C_{w,r'}$ fits into a wedge centered at w that has an angle of β at its apex v; see, for example, w_3 in Figure 4.15.

Our invariant is that, in each recursive call for $G(w')$, we have $0 \leq \alpha' < 180°$ and $r' > 0$. This ensures that our drawing has no crossings. To finish the proof, note that the segments end only in odd-degree vertices (one port each) or in degree-2 vertices (two ports each) of simple cycles that have less than three cut vertices.

Concerning the running time, it is easy to see that each recursive call of the algorithm runs in time linear in the size of the subgraph of G that the current call draws without further recursion. Hence, the overall running time is linear in the size of G (including the computation of the block-cut tree). □

Note that the algorithm in the proof of Theorem 4.22 can draw a cactus with a fixed outerplane embedding such that its embedding is maintained. Unfortunately, the drawing area can be at least exponential, even if the embedding is not fixed.

4.7 Concluding Remarks and Open Problems

For some classes of planar graphs, we have investigated the segment number from a new perspective: so far most authors asked the question *what is the minimum number of segments sufficient to draw every graph of this class crossing-free?* Instead, we have asked *what is the maximum number of segments necessary to draw every graph of this class crossing-free?* This has led us to the concept of universal lower bounds in contrast to universal upper bounds. Together with the two corresponding existential bounds, this may give us a better understanding of the range in which the segment number of a given graph can lie and what segment number we can expect from a given planar graph.

In the context of this chapter, many questions remain open. The first is to close the gaps between universal and existential bounds in Table 4.1. For the instances where we have shown universal lower bounds, we could always prove that they are tight, some up to an additive constant (this holds for maximal outerpaths, planar 3-trees, and cactus graphs). The only exceptions are maximal outerplanar graphs and 2-trees. In our proof for their universal lower bound, we have "glued" maximal outerpaths and have analyzed the maximum loss of ports per gluing operation. Our estimation of the loss is most likely not very tight (or cannot easily be made tight with this technique). With different proof techniques, however, we expect that one can prove a universal lower bound that is greater than our current number.

Of course, there are more classes and families of planar graphs to study. New results may also be more algorithmic than ours. For instance, for a planar graph G of a specific class where determining the segment number is not NP-hard, compute in polynomial time $\mathrm{seg}(G)$ and a straight-line drawing of G with $\mathrm{seg}(G)$ segments. Also other definitions of the segment number, like a 1-planar[9] version, are certainly worth being studied.

To a smaller extent, we have also investigated the arc number, that is, the minimum number of circular arcs required to draw a graph. We remark that circular arc drawings are a natural generalization of straight-line drawings. Here, even less is known than for the segment number. Also, the ratio between the segment and arc number is not yet well-understand. We have shed some first light on this question in Section 4.5. We were not able to find an example of a graph requiring more than 3 times more segments than circular arcs. This leads us to the (very) cautious conjecture that 3 is actually the upper bound. However, any argument that would just indicate that this ratio is bounded by some constant would be interesting. One might also investigate these questions in terms of computational complexity. For instance, what is the complexity of deciding whether the arc number of a given graph is strictly smaller than its segment number?

In Section 4.2, we have generalized the concept of straight-line drawings and circular-arc drawings even further and introduced the notion of pseudo-k-arcs. A pseudo-k-arc arrangement is an arrangement of curves where any pair of curves intersects at most k times. This concept is taken from pseudolines and pseudocircles, which have gained quite some attention in computation geometry – also in the context of the complexity class $\exists\mathbb{R}$. However, to the best of our knowledge, there is little work on pseudo-k-arcs (though the general concept might be well-known under different names in different disciplines). We think that they are also worth investigating in the context of the pseudo-k-arc number and beyond. Quite interestingly, the segment number and the pseudosegment number of maximal outerpaths (at least for worst-case instances) match. For drawings with circular arcs, we could not match an example we had for pseudo-circular arcs (see Proposition 4.9). This raises the question of whether we can do better with pseudo-circular arcs than with circular arcs. Also, maximal outerpaths are a relatively restricted graph class. Considering further planar graph classes is here also an interesting open topic.

[9] A drawing is *1-planar* if every edge is crossed at most once. A graph is *1-planar* if it admits a 1-planar drawing. 1-planar graphs belong to the so-called *beyond-planar* graphs that generalize planar graphs and have become increasingly popular in recent years.

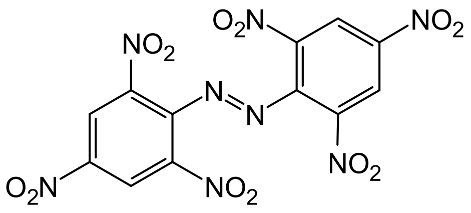
Chapter 5
Upward-Planar Drawings with Three and More Slopes

We continue our research on low visual complexity of graph drawings with straight-line segments. After analyzing the number of required straight-line segments for planar drawings of some graph classes, we now consider the number of distinct slopes in a straight-line drawing. So far, we have considered only undirected graphs and, indeed, this is also a relevant setting for determining the number of distinct slopes necessary and sufficient to draw a graph. However in this chapter, we consider upward-planar straight-line drawings, which has a long history in graph drawing specific to directed graphs (digraphs for short) or, to be more specific, to directed acyclic graphs (DAGs).

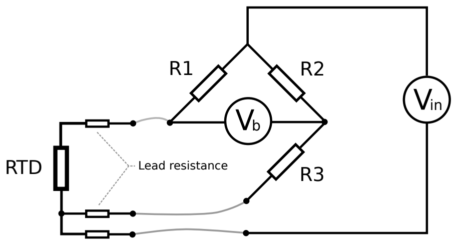
5.1 Introduction

Often, a digraph, and in particular a DAG, models hierarchical relations. In a visualization of such a hierarchy, it is natural to let all – or almost all – edges point in the same vertical direction. Interestingly, there is a split between the theoretic and the applied (or at least Sugiyama-style layer-based) graph drawing community in whether these edges should point upwards or downwards. On the one hand, within the theoretic graph drawing literature, there is a large number of publications investigating *upward* planar drawings. On the other hand, the famous framework for layered graph drawing by Sugiyama, Tagawa, and Toda [STT81] outlines an algorithm where the edges should point *downwards*. Also, later publications on layered graph drawing based thereupon assume a downward orientation of the majority of edges. In Chapter 7, we also present a layered graph drawing algorithm – however for undirected graphs. Of course, we can always mirror or rotate the drawing of a digraph, and then the actual vertical orientation of the edges does not matter. For consistency with the literature on upward graph drawing, we assume in this chapter that all edges point upwards.

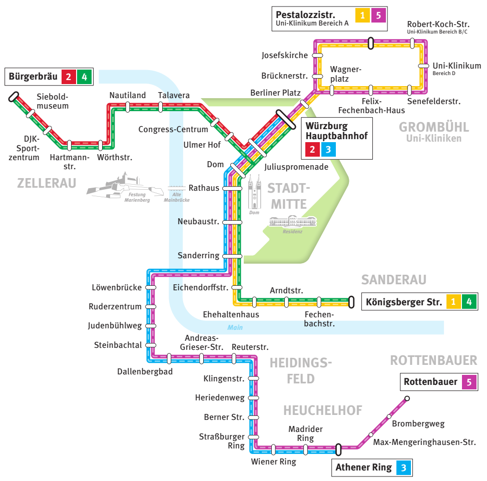
As mentioned in the previous chapter, the visual complexity is an important quality measure for the clarity and readability of a graph drawing [Sch15]. Here, we aim for a small number of distinct slopes and no crossing. Using a limited number of slopes is common for schematic drawings: Various diagram types are orthogonal drawings and use only two different slopes, namely, vertical and horizontal straight-line segments. Technical diagrams like circuit plans or cable plans (see Chapter 7) are typical examples. With three or four slopes we get hexalinear and octilinear drawings,



(a) Hexalinear schematic drawing of the explosive hexanitroazobenzene [wik21].



(b) Octilinear schematic drawing of a three-wire resistance thermometer [wik17].



(c) Octilinear schematic drawing of the tramway network of Würzburg [wvv22].

Figure 5.1: Many schematic drawings of graphs in everyday life use few discrete slopes.

respectively, which find applications in metro maps [NW11], visualization of chemical cyclic compounds [FGR04], and VLSI [MS07]. Hence, the setting where we are given a small constant number k of distinct available slopes is of high practical relevance. In Figure 5.1, we show some real-world examples for hexalinear and octilinear drawings of graphs. For an example of a generic upward-planar straight-line drawing with three slopes see Figure 5.2.

The question we investigate in this chapter is, for a given (possibly embedded) digraph G and a fixed number k of slopes, whether G admits an upward-planar straight-line drawing with k slopes. Note that the digraph G may admit an upward-planar k-slope drawing only if it has maximum in- and outdegree at most k and does not contain any directed cycle. We assume this for the remainder of this chapter.

Upward Planarity. Observe that an upward-planar embedding, given by the edge order around each vertex, is necessarily *bimodal*, that is, each cyclic sequence of incident edges around a vertex can be split into two contiguous subsequences of incoming edges and outgoing edges [DETT99]. If we are not given a bimodal embedding of a digraph G, but only G, we are required to find a (bimodal) upward-planar embedding of G (if one exists) in order to draw G upward planar.

However, Garg and Tamassia [GT01] have shown that upward-planarity testing is an NP-complete problem for general digraphs. This even holds for digraphs with bounded maximum degree Δ, where $\Delta \geq 2$ [KM22].

On the positive side, there exist several FPT-algorithms for general digraphs [Cha04, HL06, DGL10, CDF$^+$22] and polynomial-time algorithms for special graph classes such as single-source digraphs [BDMT98], outerplanar digraphs [Pap94], series-parallel digraphs [DGL10], and triconnected digraphs [BDLM94]. Moreover, if the embedding of a digraph is given, upward planarity can be tested in polynomial time as shown by Bertolazzi, Di Battista, Liotta, and Mannino [BDLM94], and once we know that a digraph G is upward planar, then we also know that G admits an upward-planar straight-line drawing due to Di Battista and Tamassia [DT88].

k-Slope Drawings. A *k-slope drawing* of a (not necessarily directed and not necessarily planar) graph G is a straight-line drawing of G where every edge is drawn with one of at most k different slopes; see Figure 5.2. Observe that the number of slopes needed to draw a graph depends on the graph. For example, the graph in Figure 5.2 does not admit a straight-line drawing with only two slopes, but, as shown there, it admits a drawing with three slopes.

Wade and Chu [WC94] define the *slope number* of G as the smallest k such that G admits a k-slope drawing. Moreover, if only planar drawings are allowed, the number is called the *planar slope number*, and if only upward-planar drawings are allowed, the number is called the *upward-planar slope number*. Both the slope number and the planar slope number have been studied extensively, mostly with the goal of finding upper bounds for particular graph classes [WC94, PP06, DSW07, DESW07, KPPT08, MS09, MP11, KPP13, LLMN13, JJK$^+$13, KMW14, DLM15, DLM18, BKM19]. We give a few examples next.

While Pach and Pálvölgyi [PP06] have proven that graphs with bounded degree Δ, where $\Delta \geq 5$, can have arbitrarily large slope number, Mukkamala and Szegedy [MS09] have shown that four slopes suffice for cubic graphs. Restricted to planar drawings, Dujmović, Eppstein, Suderman and Wood [DESW07] have shown, among other results mainly concerning the segment number (see Chapter 4), that all plane cubic graphs admit planar 3-slope drawings. In general, determining the planar slope number of a graph is hard in the existential theory of the reals ($\exists\mathbb{R}$), which was proven by Hoffmann [Hof17].

Recently, the interest in upward-planar drawings on few slopes has grown. In 2018, Bekos, Di Giacomo, Didimo, Liotta, and Montecchiani [BDD$^+$18] have shown that every so-called bitonic st-graph G with maximum degree Δ admits an upward-planar Δ-slope drawing with at most one bend per edge. A plane digraph is an st-graph if it has only one source vertex incident only to outgoing edges, only one sink vertex incident only to incoming edges, and both the source and the sink vertex lie on the outer face. An st-graph is bitonic if its vertices can be ordered in a specific way; see the work by Gronemann [Gro16] for more details. Two years later, Di Giacomo, Liotta, and Montecchiani [DLM20] have proven the same result for series-parallel digraphs. We remark that both Bekos et al. and Di Giacomo et al. use horizontally drawn segments for edges in both directions.

In 2019, Brückner, Krisam, and Mchedlidze [BKM19] have studied level-planar drawings with a fixed slope set, that is, upward-planar drawings where each vertex is drawn on a predefined integer y-coordinate (its *level* or *layer*; see also Chapter 7 in this book for layered drawings of graphs). Older works include results by Czyzowicz, Pelc, and Rival [CPR90] and Czyzowicz [Czy91] on lattices and several results for trees [CDP92, BBB$^+$08, BM10, BM13].

Previous Work. Instead of asking *how many slopes are necessary or sufficient to draw a given graph*, we turn the question around and ask *given a set of k slopes, which graphs can we draw?* This question has first been investigated by Klawitter and Mchedlidze [KM22] for $k = 2$ (arbitrary distinct) slopes. They show how to decide in linear time whether a given upward-plane digraph admits an upward-planar 2-slope drawing. In the variable-embedding scenario, they present a linear-time algorithm for single-source digraphs, a quartic-time algorithm for series-parallel digraphs, and, for general digraphs, an algorithm running in FPT-time with respect to the biconnected and triconnected components. Quapil [Qua21] recently also considered upward-planar k-slope drawings. His results include an extension of Hoffman's $\exists\mathbb{R}$-hardness from the planar slope number to the upward-planar slope number, drawings of series-parallel graphs for $k = 3$, and area requirements of upward-planar k-slope drawings for ordered trees, cacti, and series-parallel graphs.

Contribution. In this chapter, we extend the results by Klawitter and Mchedlidze [KM22] to the case of three and more slopes. More precisely, we study the problem of deciding whether a given digraph admits an upward-planar k-slope drawing for any given k with a special focus on the case $k = 3$. Broadly speaking, we show that the problem becomes harder with the complexity of the graph class.

First, we consider ordered and unordered directed trees (defined below) as a warm-up to our problem (Section 5.3). In particular, the upward-planar slope number of a directed tree is easy to determine and an upward-planar k-slope drawing can be constructed efficiently. Second, we show that for a given cactus digraph G, we can construct an upward-planar k-slope drawing in polynomial time (Section 5.4). To this end, we devise a dynamic program on the block-cut tree of G and utilize a simple polygon drawing algorithm by Culberson and Rawlins [CR85]. As a third efficiently drawable graph class (but only for $k = 3$ slopes), we consider inner triangulations (Section 5.5), which also contain maximal outerplanar digraphs and maximal outerpaths.

In the second half of this chapter, we enter the realm of NP-hardness. We start with an NP-hardness construction for upward-outerplanar digraphs and $k = 3$ slopes (Section 5.6), onto which all of the following sections are based. This construction is applicable in both the fixed- and the variable-embedding scenario. Afterwards, we strengthen this result to upward-planar outerpaths (Section 5.7). Finally, we consider $k \geq 4$ slopes. There, we show NP-hardness for upward-planar digraphs (Section 5.8) – in the fixed-embedding scenario for all $k \geq 4$ and in the variable-embedding scenario for $k \geq 5$. We also argue why we leave the case $k = 4$ open.

Prior to that, in the next section, we specify the setting for the slopes in an upward-planar drawing. Our findings and some known results are summarized in Table 5.1.

graph class	embedding	$k = 2$	$k = 3$	$k = 4$	$k \geq 5$
tree	fixed	P [KM22]	P C5.4	P C5.4	P C5.4
	variable	P T5.1	P T5.1	P T5.1	P T5.1
cactus	fixed	P [KM22]	P T5.5	P T5.5	FPT T5.5
	variable	P T5.5	P T5.5	P T5.5	FPT T5.5
inner triang.	fixed	P O5.6	P T5.9	?	?
	variable	P O5.6	P T5.11	?	?
outerpath	fixed	P [KM22]	NPh T5.15	?	?
	variable	FPT [KM22]	NPh C5.16	?	?
outerplanar	fixed	P [KM22]	NPh T5.13	?	?
	variable	FPT [KM22]	NPh C5.14	?	?
planar	fixed	P [KM22]	NPh T5.13	NPh C5.17	NPh C5.17
	variable	FPT [KM22]	NPh C5.14	?	NPh T5.18

Table 5.1: Complexity of computing an upward-planar k-slope drawing of a given digraph of some graph classes. P means polynomial-time solvable, FPT means fixed-parameter tractable, and NPh means NP-hard. Our findings (blue) refer to theorems (T), corollaries (C), and observations (O).

5.2 Preliminaries

Before we investigate the question whether a digraph of some graph class admits an upward-planar k-slope drawing, we need to clarify what it actually means to *admit an upward-planar k-slope drawing.*

First, note that we consider both the fixed- and variable-embedding scenario. When we are in the fixed-embedding scenario, we assume that we are given a planar embedding with the characteristic property of that graph class, that is, an upward-outerplanar embedding for an upward-outerplanar digraph, an upward-outerpath embedding for an upward-outerpath and so on. Of course, we then require that the input embedding is preserved. On the other hand, if we are in the variable-embedding scenario, then we ask the question of whether the input digraph admits an upward-planar k-slope drawing with the characteristic property of the input graph class. For example, if we are given a directed outerpath G, the task is to decide whether there is an upward-outerpath k-slope drawing of G.

Second, we have not yet spoken about *which* slopes to use in a drawing. It is at least not obvious if one should expect different answers to the realizability question depending on the set of the k slopes. Also, it is not clear whether the set of k slopes is part of the input or can be chosen by an algorithm. Then, only the number k would be part of the input (or there is a specific algorithm only for a specific k).

Slope Sets. Of course, we need a set of k distinct slopes. To obtain a slope set, we propose the following three settings, which are illustrated in Figures 5.2b to 5.2d.

general setting: Any set of k distinct slopes can be chosen. This choice may be up to the user or the algorithm.

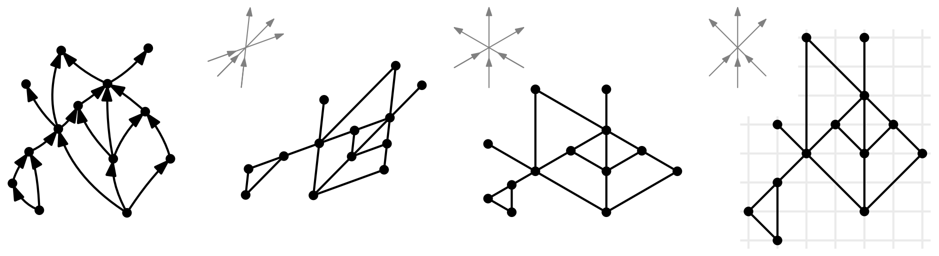

(a) Digraph G. (b) General setting. (c) Uniform-angles setting. (d) Regular-grid setting.

Figure 5.2: The upward-planar straight-line line drawings of the digraph G realize different slope sets. In the remained of this chapter, edges are drawn upward while arrow heads are omitted.

uniform-angles setting: The slopes are distributed equally, that is, clockwise from the x-axis they have angles in $\{i \cdot \pi/k - \frac{\pi}{2k} : i \in \{1, \dots, k\}\}$.

regular-grid setting: Any set of k distinct slopes that connect points on the 2D grid is allowed.[10] This may include the horizontal slope and then horizontally rightwards also counts as upwards. We prefer to pick slopes that can be used by segments of roughly equal length; see for example Figure 5.2d and Figure 5.4.

Both the uniform angles and the regular-grid setting have their own advantages and disadvantages. Uniform angles naturally lead to more balanced drawings with more rotational symmetry, which we find more visually appealing. Moreover, the drawings have a perfect angular resolution, which is also known as a quality measure of graph drawings [DETT99, DLM18]. The downside of the uniform-angles setting is that we cannot always use grid points of the regular 2D grid. For example, for $k = 3$, the first slope is $\tan(\pi/6) = 1/\sqrt{3}$, which is an irrational number. Henceforth, we assume for uniform angles a computation and representation model that can handle implicit coordinates or alternatively real numbers. On the other hand, while the regular-grid setting naturally facilitates integer coordinates, it may also yield less balanced angles between edges and irrational edge lengths. Since all of these settings have their natural justification, we do not restrict our considerations to one of them.

Note that a 2-slope drawing can be sheared such that one slope changes and the other remains the same; see Figure 5.3ii; we refer for further explanations to Klawitter and Mchedlidze [KM22]. Moreover, observe that such a transformation only changes the lengths of the segments of the changed slope, but not the lengths of the other segments. The fact that this does not hold for three or more slopes introduces interesting new geometric aspects for these cases. However, note that $k = 3$ is a special case because no matter which three slopes we pick, they can be affinely transformed to the slopes of the angles $\{45°, 90°, 135°\} = \{\nwarrow, \uparrow, \nearrow\}$, as illustrated in Figure 5.3. This set of slopes belongs to the regular-grid setting. Hence, when referring to $k = 3$, we assume the use of this slope set unless stated differently.

[10] We require that the distance between two such points on the 2D grid is in a sense constant and does not depend on the size of the input (graph).

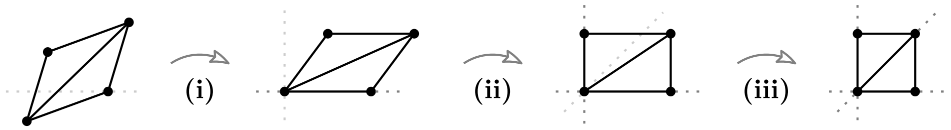


Figure 5.3: Given a drawing using a set of any three slopes, we can (i) rotate, (ii) shear, and (iii) stretch it to a drawing with the slope set $\{\nwarrow, \uparrow, \nearrow\}$ (rotated here by 45° clockwise for illustrative purposes to $\{\uparrow, \nearrow, \rightarrow\}$).

For illustrative purposes, we rotate drawings in Section 5.6 by 45° clockwise and then use the slope set $\{\uparrow, \nearrow, \rightarrow\}$. In some cases, we number k different slopes with the numbers 1 to k in counterclockwise order.

Slope Assignment. A *k-slope assignment* of a digraph G assigns each edge of G one of k slopes. If G is upward plane, we call a k-slope assignment of G *consistent* if the assignment complies with the cyclic edge order around each vertex; that is, for $k = 3$, if a vertex has three incoming edges, they need to be assigned the slopes $\nearrow$, $\uparrow$, and $\nwarrow$ in counterclockwise order. Clearly, if an upward-plane embedding does not admit a consistent k-slope assignment, then it also does not admit an upward-planar k-slope drawing.

5.3 Directed Trees

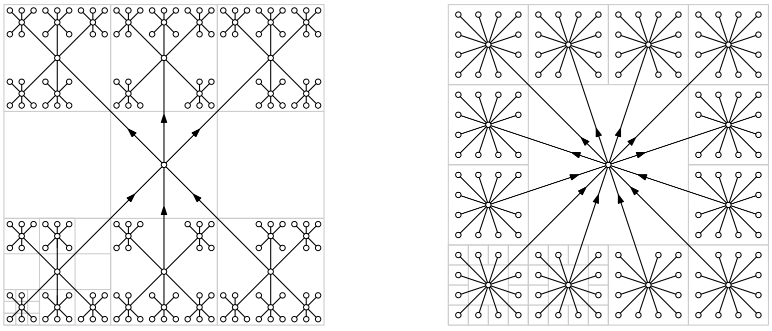
In this section, we consider upward-planar k-slope drawings of directed trees. While our trees are in general not rooted, results for rooted trees can be derived or are partially already known [BM10, BM13]. Note that naturally, every unordered tree is upward planar, while an ordered tree is upward planar if and only if its embedding is bimodal.

5.3.1 Unordered Trees

The planar slope number of an unordered undirected tree with maximum degree Δ is $\lceil \Delta/2 \rceil$ [DESW07]. Intuitively, this is relatively easy to see: imagine a recursive approach adding leaves to the existing drawing, where we can attach to each vertex per slope two edges.

Therefore, it is not surprising that the upward-planar slope number of an unordered directed tree T equals the maximum of the largest indegree of T and the largest outdegree of T. To show this, we draw T as subgraph of a larger, regular tree $T_{k,h}$ for $h \geq 1$ where every non-leaf vertex has in- and outdegree k and each leaf has distance h to a central vertex.[11] To draw $T_{k,h}$ on a grid with k slopes, we adopt the strategy of Bachmaier, Brandenburg, Brunner, Hofmeier, Matzeder, and Unfried [BBB+08]

[11] We can imagine this central vertex as the root of the tree. However, we avoid the term *root* here because the edges are not all directed away from that root. There is no global source vertex with respect to the edge directions.



(a) $T_{3,3}$, $k = 3$. **(b)** $T_{6,2}$, $k = 6$.

Figure 5.4: Upward-planar k-slope drawings of unordered trees on the grid.

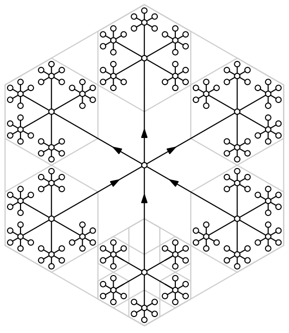
from 2008 for complete rooted trees; see Figure 5.4. It works within the regular-grid setting. Alternatively, $T_{k,h}$ can be drawn with k uniform angles; see Figure 5.5.

Theorem 5.1. *Let T be an unordered directed tree on n vertices with maximum indegree and outdegree at most k. Then T admits an upward-planar k-slope drawing in the regular-grid setting and T admits an upward-planar k-slope drawing in the uniform-angles setting. Moreover, the drawings can be computed in $\mathcal{O}(n)$ time.*

Proof. Let ℓ be the number of vertices on a longest path in the underlying undirected graph $U(T)$ of T. Since $U(T)$ does not contain cycles, ℓ is well-defined. We first describe how to construct an upward-planar k-slope drawing of $T' = T_{k,\lceil \ell/2 \rceil}$. Let ρ be the central vertex of T'. Since T is a subgraph of T', a drawing for T could be obtained from the drawing of T' straightforwardly. However, since T' might be substantially larger than T, we describe at the end how to construct the drawing of T directly, which allows for linear runtime.

To draw $T_{k,\lceil \ell/2 \rceil}$ on a grid, we first place ρ at the center of an axis-aligned square that represents the drawing region. We then partition this larger square into $(\lceil k/2 \rceil + 1)^2$ smaller equal-sized squares. The neighbors of ρ are placed at the centers of the smaller squares that appear along the perimeter of the larger square. If k is even, then all of these smaller squares get occupied where the left central square gets a predecessor of ρ and the right central square gets a successor of ρ. If k is odd, then the left and right central squares remain unoccupied. For each neighbor v of ρ, we then proceed recursively within the square of v; see Figure 5.4. Hence, the trees obtained from removing ρ in T' are all drawn in disjoint regions and are thus non-overlapping. The recursive procedure ends at the leaves and with the smallest squares. The size of these smallest squares determines the final grid size.

Alternatively, to draw T' with k slopes and uniform angles, we use a regular $2k$-gon (instead of a square) as drawing region and place ρ at its center. For the subtrees, we then use appropriately smaller regular $2k$-gons; see Figure 5.5. More precisely, we pick the sizes such that one corner of each smaller $2k$-gon is incident to a corner of the larger $2k$-gon, and, for odd k, two small $2k$-gons touch at a corner, as


(a) Unordered tree $T_{3,3}$, $k = 3$.

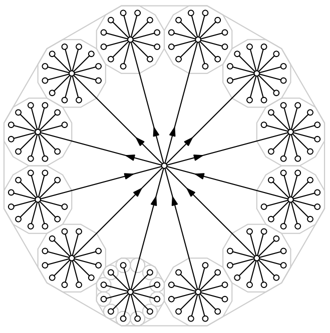

(b) Unordered tree $T_{6,2}$, $k = 6$.

Figure 5.5: Upward-planar k-slope drawings of unordered trees with uniform angles.

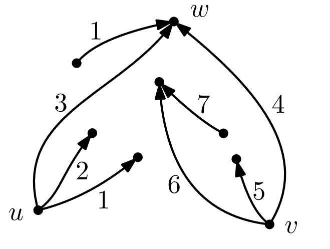
in Figure 5.5a, while for even k, they touch at a side, as in Figure 5.5b. The process continues recursively as before.

It remains to describe how to compute such drawings for T directly and obtain linear runtime. First, find a longest path P in $U(T)$ in linear time using the algorithm by Bulterman et al. [BvdSZ⁺02] and determine a vertex ρ in the middle of P. The middle vertex of P is unique in case P has even length and we can arbitrarily choose one of the two middle vertices otherwise. Second, recursively construct a consistent k-slope assignment of T with ρ as a start vertex. Clearly, assigning the slopes greedily yields a consistent k-slope assignment. For the next step, note that the smaller squares or $2k$-gons are only used to find (and prove) appropriate vertex positions for each depth. However, we can also compute these positions (and the grid size) directly for each vertex of T in constant time. Finally and again recursively starting from ρ, draw the edges of T with their designated slopes such that they terminate at the computed vertex positions. Overall, each step and thus the whole algorithm runs in $\mathcal{O}(n)$ time. □

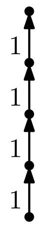
Note that the recursive drawing procedure from Theorem 5.1 requires exponential-size drawing area in the length ℓ of the longest path in $U(T)$ (or at least an exponential edge-length ratio). For an arbitrary number of slopes, Frati [Fra08] showed that an area of size $\Theta(n \log n)$ suffices to draw an n-vertex tree upward planar. As far as we know, it remains open whether upward-planar k-slope drawings of unordered directed trees require exponential area. This is the case for ordered directed trees, which we consider next.

5.3.2 Ordered Trees

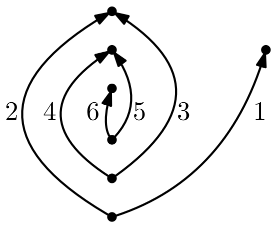
As stated before, the ordered directed trees that admit an upward-planar drawing are precisely the ordered directed trees whose embedding is bimodal. (Clearly, an upward-planar embedding is bimodal and if we have a bimodal embedding, we can find a recursive drawing approach similar to the one from Theorem 5.1 but using arbitrary slopes.)



(a) The third incoming edge vw of w cannot have slope 3.



(b) A directed path with upward-planar slope number 1.



(c) A directed path with upward-planar slope number $n-1$.

Figure 5.6: The upward-planar slope number of an n-vertex ordered directed tree lies between 1 and $n-1$, since it is not determined locally. Here the number represents different slopes.

Let T be an ordered directed tree in a bimodal embedding. To determine the upward-planar slope number of T, it suffices to find a consistent k-slope assignment for T with minimum k. We can then use the drawing algorithm for unordered trees from Theorem 5.1. In this regard, note that the maximum in- and outdegree are natural lower bounds but that the choice of the (minimal) slope for an edge vw cannot be determined locally at v and w. For example, the edge vw in Figure 5.6a is the third incoming edge at w but requires at least slope 4, since its preceding edge uw already requires slope 3 at u. This effect only appears along alternating paths of incoming and outgoing edges. Hence we have the following observation; see also Figures 5.6b and 5.6c.

Observation 5.2. *The upward-planar slope number of ordered directed trees with n vertices ($n \geq 2$) is bounded within 1 and $n-1$ and these bounds are tight.*

A consistent k-slope assignment for T that minimizes k can be constructed with a simple greedy algorithm in linear time.

Theorem 5.3. *The upward-planar slope number k of an ordered directed tree on n vertices can be determined in $\mathcal{O}(n)$ time. Moreover, an upward-planar k-slope drawing of T can be constructed in $\mathcal{O}(n)$ time.*

Proof. Let T be an ordered directed tree on n vertices. We describe a greedy algorithm that computes a consistent k-slope assignment for T where k is minimum. The idea is that we assign the slopes $1, 2, \ldots$ in counterclockwise (ccw) order around each vertex, but an edge uv receives a slope only if all outgoing edges at u that ccw precede uv and all incoming edges at v that ccw precede uv already got a slope assigned. Therefore, an edge uv receives a mark when it has this property at u or at v, and gets added to a queue as soon as it receives its second mark. To find the first edge in the queue, we mark the ccw first incoming and outgoing edge at each vertex; this can be done in overall $\mathcal{O}(n)$ time. After this initialization, there always exists an edge with two marks by the pigeonhole principle: for every vertex, we have added at least one mark and there is one edge less than there are vertices in a tree.

For an edge uv that we take from the head of the queue, we assign uv the lowest available slope, that is, one plus the maximum of the slope of its ccw preceding outgoing edge at u and its ccw preceding incoming edge at v; if there is no preceding edge, then slope 1 is available. We then mark the ccw succeeding edges of uv at u and v. Since T is a tree, there is (until the algorithm terminates) always at least one edge in the queue by a similar argument as before.

When the algorithm terminates, each edge is assigned a slope, which increases in ccw order for the incoming and outgoing edges of each vertex. Furthermore, the highest assigned slope is also the upward-planar slope number of T with respect to its embedding. The running time of the algorithm is in $\mathcal{O}(n)$ since the algorithm considers each edge only a constant number of times.

Once we have computed the upward-planar slope number k of T with respect to its embedding, we can use the drawing algorithm from Theorem 5.1 to compute an upward-planar k-slope drawing of T again in $\mathcal{O}(n)$ time. □

Corollary 5.4. *Let T be an ordered directed tree on n vertices with maximum indegree and outdegree at most k. We can decide in $\mathcal{O}(n)$ time whether T admits an upward-planar k-slope drawing.*

Frati [Fra08] showed that an alternating ordered directed path on n-vertices as in Figure 5.6c requires $\Omega(2^n)$ area. However, this path also requires $n-1$ slopes. It is hence natural to ask whether there exist also ordered directed trees that require an exponential area in upward-planar k-slopes drawings for a constant k. This question has recently been answered by Quapil [Qua21] for $k = 3$ by constructing a similar path with a spiral structure that requires exponential area. We remark that his construction can also be extended to $k > 3$.

5.4 Cactus Digraphs

In this section, we show that deciding whether a given cactus digraph admits an upward-planar k-slope drawing is fixed-parameter tractable (FPT) in k. This holds for both, the fixed- and the variable-embedding scenario. In particular, constructing an upward-planar k-slope drawing of a cactus digraph can be done in quadratic time if k is constant. We remark that our algorithm only works for the uniform-angles setting. For illustrative purposes and $k = 3$, we also use the slopes $\{\nwarrow, \uparrow, \nearrow\}$.

Note that an acyclic cactus digraph is always upward planar and if no upward-planar embedding is specified, then the algorithm can compute one. However, as for ordered directed trees, there may be embedded cactus digraphs without upward-planar k-slope drawing. Unlike directed trees, also in the variable-embedding scenario not every cactus digraph with maximum in- and outdegree at most k admits an upward-planar k-slope drawing as noted by Quapil [Qua21]; see Figure 5.7.

Our FPT-algorithm consists of a preprocessing and three phases. In the preprocessing, we compute the blocks and the block-cut tree of the input cactus G. In the first phase, we compute a combinatorial realization for each block. Roughly speaking, we use a dynamic program on the block-cut tree of G that computes

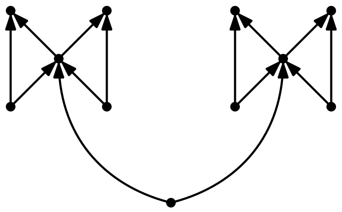
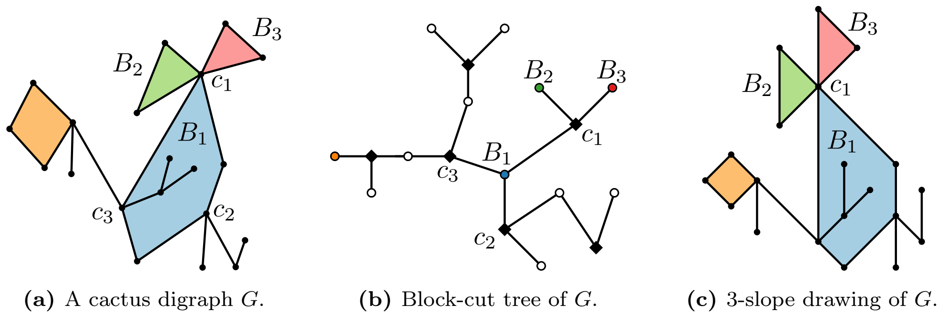


Figure 5.7: A Cactus digraph with maximum in- and outdegree 3, which, even in the variable-embedding scenario, does not admit an upward-planar 3-slope drawing as shown by Quapil [Qua21].



(a) A cactus digraph G. **(b)** Block-cut tree of G. **(c)** 3-slope drawing of G.

Figure 5.8: The nodes (blocks) of the block-cut tree of a cactus digraph G correspond to the cycles and edges of G; we draw all blocks separately and then merge their drawings.

combinable k-slope assignments for each block. In the second phase, we compute a geometric realization for each block and finally, in the third phase, we combine the geometric realizations of the blocks to a drawing of the input cactus.

For a cactus digraph G to admit an upward-planar k-slope drawing, each block of $\mathcal{T}$ must be drawable under constraints imposed by other blocks. Consider for an example the cactus G in Figures 5.8a to 5.8c under the given embedding. For $k = 3$, the two edges of the block B_1 incident to the cut vertex c_1 need to get the slopes $\uparrow$ and $\nwarrow$ because of the blocks B_2 and B_3. Our strategy is thus as follows.

Algorithm. In the first phase, we run a dynamic program (described below) on the blocks of $\mathcal{T}$ to find a consistent slope assignment for each block such that the blocks are *combinatorially* combinable. If successful, we enter the second phase, where we compute drawings of the blocks that are *geometrically* combinable. In the last phase we put all block drawings together.

Let G be a cactus digraph with blocks $B_1, \dots, B_\ell$ and let $\mathcal{T}$ be the block-cut tree of G. We pick an arbitrary block node, say B', of $\mathcal{T}$ as root and direct all edges of $\mathcal{T}$ towards B'. As a result, each block node B (except B') has exactly one outgoing edge towards a cut vertex c in $\mathcal{T}$. We then say c is the *anchor* of B. Let B be a cycle block with anchor c and suppose we have a slope assignment for B. Let e and e' be the edges of B incident to c such that if we move clockwise (cw) along the inner face of B, we have the sequence e, c, e'. Then the *anchor type* $t_c(B)$ of c for B is

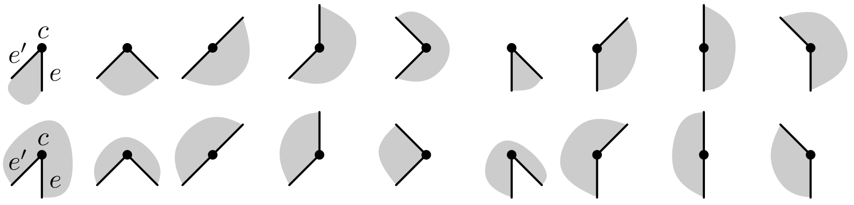


Figure 5.9: A subset of the anchor types of a cycle block for $k = 3$.

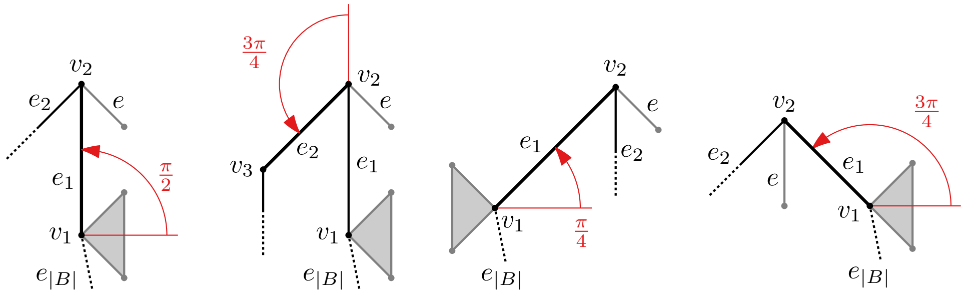
defined as the slopes of e and e' and if they are incoming or outgoing edges at c; see Figure 5.9. For an edge block B with edge e, the *anchor type* $t_c(B)$ describes the slope of e and if e is incoming or outgoing at c. For cycle blocks and edge blocks, there are $2k \cdot (2k-1)$ and $2k$ different anchor types, respectively.

For a block node B with anchor c, a *block tuple* $\tau_B = \langle \phi_B, t_c(B) \rangle$ consists of a consistent k-slope assignment ϕ_B of B and an anchor type $t_c(B)$ of c. A block tuple τ_B is *feasible* if B has no descendant blocks or if B's descendant blocks admit a non-empty set of feasible block tuples that can be combined with τ_B. A *feasible set* for B is a maximal set of feasible block tuples for B that have pairwise different anchor types. We process $\mathcal{T}$ in a post-order traversal. This means, we first process the children nodes of a block node B and afterwards, we process B. For each block, we compute a feasible set based on the feasible sets of its descendant blocks.

Combinatorial Realization. Computing the feasible set of a cycle block B with anchor c works as follows. Let B be the cycle $(c = v_1, e_1, v_2, e_2, \dots, v_{|B|}, e_{|B|}, v_1)$ – if an embedding is given, let this order be cw around the inner face.

In a dynamic programming approach, we find for each edge e_i with $i \in \{1, \dots, |B|\}$ all possible slopes of e_i based on the possible slopes of e_{i-1}. Furthermore, we also consider how far we have rotated along the boundary of B from v_1 to v_{i+1} to keep track of whether we have a total sum of inner (outer) angles around B of $\pm 2\pi$. More precisely, we walk around B once and store for each edge e_i, $i \in \{1, \dots, |B|\}$, all tuples of (i) a possible slope s of e_i and (ii) the corresponding sum α of rotation angles when traversing B from v_1 to v_{i+1} (consistently, along one side of B). We call the tuple $\langle s, \alpha \rangle$ an *edge tuple* of e_i. As a base for the sum of rotation angles, when considering e_1, we use the angle a horizontal ray emanating at v_1 and pointing towards positive infinity would need to rotate to contain e_1; see Figure 5.10a.

Next, we describe how to compute all edge tuples of the edges of a block B. We start with computing the edge tuples of e_1 by considering all combinations of feasible block tuples of descendant blocks of B anchored at v_1 and v_2. For each possible combination, we can test which slopes e_1 may get in $\mathcal{O}(k)$ time. The corresponding rotation angle can be computed in $\mathcal{O}(1)$ time. A descendant cycle block can have $\mathcal{O}(k^2)$ many feasible block tuples, while a descendant edge block can have $\mathcal{O}(k)$ many feasible block tuples. Together at v_1 and v_2, there can be at most $2(k-1)$ descendant cycle blocks and at most $4(k-1)$ descendant edges. The total number of possible combinations of feasible block tuples of descendant blocks of B is thus at most



(a) For a fixed embedding, e_1 starts with slope $\uparrow$, ...

(b) ... and then e_2 gets slope $\nearrow$.

(c) In the variable-embedding scenario, e_1 may also get slope $\nearrow$ ...

(d) ... or e_1 may get slope $\nwarrow$.

Figure 5.10: Computing a slope assignment for a cycle block B with anchor v_1. The algorithm handles the edges of B one by one starting with e_1.

$\mathcal{O}((k^2)^{2k-2}) = \mathcal{O}(k^{4k-4})$. (To see this, note that pairs of edge block descendants can be considered as cycle block descendants for this calculation.) Therefore, we can find all possible edge tuples of e_1 in $\mathcal{O}(k \cdot k^{4k-4}) = \mathcal{O}(k^{4k-3})$ time. In the example of Figure 5.10a for $k = 3$, assuming a fixed embedding, the edge e_1 can only have slope $\uparrow$ and we have thus rotated $\pi/2$. For this edge tuple $\langle\uparrow, \pi/2\rangle$, the edge e_2 in Figure 5.10b has also only one possible slope, namely $\nearrow$, and the rotation increases by $3\pi/4$ to a total of $5\pi/4$. However, in the variable-embedding scenario, e_1 can also have slopes $\nearrow$ and $\nwarrow$, see Figures 5.10c and 5.10d.

In general, to compute the edge tuples of an edge e_i for $i \in \{2, \ldots, |B|\}$, we can use the same procedure to determine all possible slopes of e_i as we have described for e_1. The main difference is that we need to take into account also all edge tuples of e_{i-1}. We can treat the slope of an edge tuple of e_{i-1} like another descendant edge block. Additionally, we update the sum of rotation angles for e_i. The number of different edge tuples of e_{i-1} is in $\mathcal{O}(k|B|)$ because each edge tuple of e_{i-1} has one of k slopes, the sum of angles for each pair of edge tuples having the same slope differs by a multiple of 2π, and each of the $\mathcal{O}(|B|)$ previous edges adds a rotation of $< 2\pi$. Thus, since we have $\mathcal{O}(k|B|)$ edge tuples for e_{i-1}, $\mathcal{O}(k^{4k-4})$ feasible block tuples of descendant blocks, and $\mathcal{O}(k)$ possible slopes for e_i, we can compute all edge tuples of e_i in $\mathcal{O}(k^{4k-2}|B|)$ time. For all edges of B this takes $\mathcal{O}(k^{4k-2}|B|^2)$ time.

Note that we may obtain the same edge tuple in multiple ways – the actual number of edge tuples for each e_i is again $\mathcal{O}(k|B|)$. For each edge tuple t of e_i, we store a pointer to t at the $\mathcal{O}(k)$ edge tuple(s) of e_{i-1} that t is based on. In other words, we build a digraph H on the edge tuples of the edges $e_1, \ldots, e_{|B|}$, where the edge tuples of e_1 are the source nodes. The digraph H has $\mathcal{O}(k|B|^2)$ vertices and $\mathcal{O}(k^2|B|^2)$ edges.

When we handle $e_{|B|}$, we discard all tuples that do not result in a 2π rotation if the embedding is given or in a $\pm 2\pi$ rotation if no embedding is given. This ensures that the cycle has a geometric realization [CR85]. To determine the feasible set for B, we start a breadth-first search (BFS) on H for each edge tuple of e_1. For each edge tuple of $e_{|B|}$ we reach, we combine the slope of e_1 and $e_{|B|}$ as well as whether the rotation is $+2\pi$ or -2π to get an anchor type of B at c. By backtracking from the edge tuple of $e_{|B|}$ within the BFS tree, we find a consistent slope assignment of B, which yields a feasible block tuple.

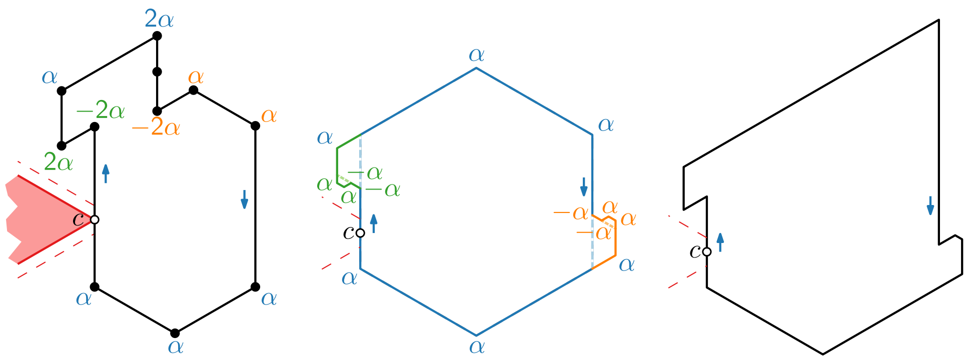
We have $\mathcal{O}(k)$ BFS runs, which take $\mathcal{O}(k^2|B|^2)$ time each and $\mathcal{O}(k^3|B|^2)$ time in total. Backtracking for all $\mathcal{O}(k^2)$ anchor types is in $\mathcal{O}(k^2|B|)$ time in total. Hence, these steps are dominated by the computation of the edge tuples and we can compute a feasible set for B in $\mathcal{O}(k^{4k-2}|B|^2)$ time. Over all blocks of G, this is in $\mathcal{O}(k^{4k}n^2)$ since each vertex is in at most k blocks and, therefore, we have $\sum_{i=1}^{\ell}|B_i|^2 \leq (kn)^2$.

Finally, let us describe how to obtain a combinatorial realization for G. If the feasible set of the root block B' is non-empty, then there is a combinatorial realization of G. To find one, we pick any feasible block tuple of B' and try to combine it with the feasible block tuples of its descendant blocks. (We know that a consistent combination exists.) Since each feasible set has size $\mathcal{O}(k^2)$, we can find a consistent set of feasible block tuples of descendant blocks in $\mathcal{O}((k^2)^{k-1})$ time per vertex of B'. Over all vertices of G this is in $\mathcal{O}(k^{2k-2}n)$ time.

Geometric Realization. Suppose that we have found a combinatorial realization in the form of a consistent k-slope assignment for every cycle and edge of G. In the variable-embedding scenario, we now know whether and how cycles nest. We thus re-root $\mathcal{T}$ such that the root block lies on the outer face. In the following, we describe how to obtain a drawing of a cycle block B as a polygon that does not intersect the edges of its parent block B' at its anchor point c.

We describe this only for the uniform-angles setting and leave it as an open question for the regular-grid setting. Given any sequence σ of rational angles (i.e., a rational number times π) that sum up to $\pm 2\pi$, Culberson and Rawlins [CR85] describe an algorithm that outputs a polygon with σ as turning angles. Their so-called *Turtlegon* algorithm works as follows. It defines a base angle α as the greatest common divisor of all angles in σ; in our case this is π/k (w.l.o.g. we can assume that our drawings are slightly rotated by $\frac{\pi}{2k}$). Larger angles are split into sequences of $\pm\alpha$ angles resulting in a new angle sequence σ'. W.l.o.g. let σ' contain more angles $+\alpha$ than $-\alpha$. Using some of the αs, the Turtlegon algorithm draws a regular $(2\pi/\alpha)$-gon (in our case $2k$-gon). To accommodate additional angles in between, it inserts exponentially shrinking detours at the corners of the $(2\pi/\alpha)$-gon; see Figure 5.11b. In the end, we get the original larger angles from merging the smaller angles [CR85]; see Figure 5.11c.

The difficulty for us, when employing this $\mathcal{O}(k|B|)$-time algorithm, is to ensure that the edges of the parent block B' can reach the anchor point c without intersecting the polygon of B. This might be impossible if c lies within a spiral inside a detour. However, we can avoid this if we let an incident edge of c be a side of the $2k$-gon (this is always possible because we can pick an appropriate set of α angles of σ' for the $2k$-gon) and if we let each detour edge shrink by a sufficiently large factor (e.g., $k|B|$); see Figure 5.11.



(a) A combinatorial description of a block B with angle sequence that contains large angles ...

(b) ... is converted to an angle sequence on base angle α and realized as $2k$-gon with detours.

(c) Small angles originating from large angles are merged back together.

Figure 5.11: The Turtlegon algorithm by Culberson and Rawlins [CR85] realizes an angle sequence of a block B with anchor point c by first splitting large angles, constructing a $2k$-gon with exponentially shrinking detours, and then merging small angles back together. When using the Turtlegon algorithm, we ensure that the anchor point c lies at a $2k$-gon edge; here $k = 3$.

The running time of this step is in $\mathcal{O}(k|B|)$. Since each vertex is in at most k blocks, we have that $\sum_{i=1}^{\ell}|B_i| \leq kn$ and, hence, the total running time is in $\mathcal{O}(k^2n)$.

Putting Blocks Together. We start with a drawing of the root block. We then recursively draw each child (in a BFS-like order) such that its anchor point coincides with the corresponding vertex of the parent polygon and scale down the drawing of the child block such that the appended polygon does not intersect the existing drawing. Note that it always suffices to scale down each child to the size of the minimum distance of any two vertices within in the existing drawing. We can determine vertex pairs of minimum and maximum distance in $\mathcal{O}(|n| \log |n|)$ time and then place and scale each polygon of a block B in $\mathcal{O}(|B|)$ time.

The total running time is dominated by the dynamic program and is thus in $\mathcal{O}(k^{4k}n^2)$. For constant k, this is a quadratic-time algorithm.

Theorem 5.5. *Let G be an upward-planar (or plane) cactus digraph with maximum in- and outdegree at most k. It can be constructively tested in $\mathcal{O}(k^{4k}n^2)$ time whether G admits an upward-planar k-slope drawing in the uniform-angles setting. In other words, constructing an upward-planar k-slope drawing of a cactus digraph is fixed-parameter tractable in k.*

For the regular-grid setting, we cannot use the algorithm by Culberson and Rawlins [CR85] because we have irrational multiples of π as turning angles. For a sequence of general turning angles, the algorithm by Hartley [Har89] computes a polygon realizing that sequence. However, it is not immediately clear how to guarantee that the edges of the parent polygon at the anchor point are not intersected. For

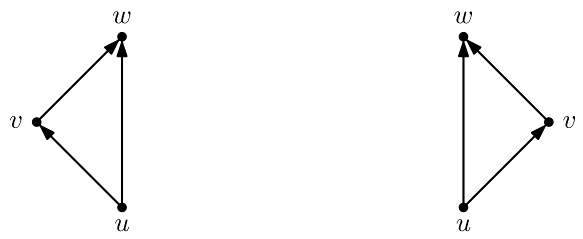


Figure 5.12: The only two ways to draw a directed C_3 with three slopes up to scaling.

general polygons, we believe that we can iteratively shrink the spikes to resolve potential intersections. Since such a procedure involves some more technicalities, we leave it as an open question.

5.5 Inner Triangulations

In this section, we focus purely on the case $k = 3$ and exploit the special structure for directed triangles there. Our findings are based on the following observations.

First, we observe that a triangle in an upward-planar digraph has necessarily a source vertex with two outgoing edges and necessarily a sink vertex with two incoming edges if we ignore the rest of the digraph. Otherwise, it would have a directed cycle, which contradicts upward planarity.

Second, we observe that we cannot draw a directed triangle with only two slopes.

Observation 5.6. *For two slopes, there cannot be any directed C_3 in an upward-planar drawing.*

Proof. Assume for a contradiction that we have a directed triangle $\langle u, v, w\rangle$ drawn with only two slopes. Then, at the source vertex u, the two outgoing edges have slopes 1 and 2. Without loss of generality, assume that the edge with slope 1 terminates at the sink vertex w. This means that the second edge at w has slope 2 and, hence, the third vertex v of the triangle has an incoming and an outgoing edge both with slope 2. Then, however, the lines with slopes 1 and 2 through u intersect a second time at w with $u \neq w$, which is not possible in Euclidean geometry. □

Third, we observe that the drawing of a directed triangle $\langle u, v, w\rangle$ is unique up to scaling and mirroring when we have three slopes.

Observation 5.7. *For three slopes, a directed C_3 in an upward-planar drawing can only be drawn as in Figure 5.12: every slope appears exactly once and the middle slope is used by a long edge from the source vertex to the sink vertex. Once we have set the length of one of the edges, the lengths of the other two edges are fixed.*

Proof. Again consider a triangle $\langle u, v, w\rangle$. The edge uw from the source u to the sink w needs to have the middle slope $\uparrow$. Only then, the path of length two from u to w through v can be drawn: the first edge with slope $\nwarrow$ and the second edge

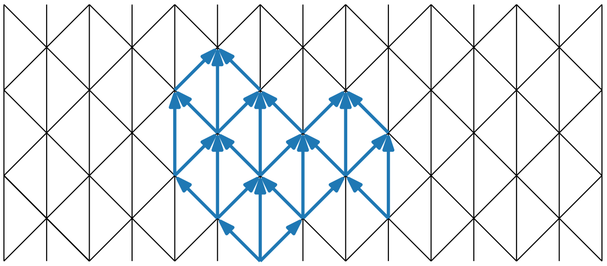


Figure 5.13: Infinite triangular grid over the slopes $\{\nwarrow, \uparrow, \nearrow\}$ and an upward-planar 3-slope drawing of an inner triangulation (blue). Any upward-planar 3-slope drawing of an inner triangulation can be scaled and translated to be a subgraph of this grid.

with slope $\nearrow$ or the first edge with slope $\nearrow$ and the second edge with slope $\nwarrow$; see Figure 5.12. Clearly, for the slope set $\{\nwarrow, \uparrow, \nearrow\}$, the edges uv and vw have equal length, which is shorter than the length of uw, and these lengths depend on each other. The length ratio between them is $1 : \sqrt{2}$ by the Pythagorean theorem. □

Fourth, we observe that, when we attach multiple equal-size triangles, we obtain a regular triangular grid.

Observation 5.8. *Any upward-planar 3-slope drawing of an inner triangulation is a subgraph of the infinite triangular grid depicted in Figure 5.13.*

Proof. When we attach multiple triangles from Figure 5.12, they all need to have the same size due to Observation 5.7. Therefore, we obtain a regular triangular grid as depicted in Figure 5.13. Consequently, any upward-planar 3-slope drawing of an inner triangulation is a subgraph of this infinite triangular grid. □

Based on Observations 5.7 and 5.8, we derive a simple algorithm for drawing a plane inner triangulation upward planar with three slopes if such a drawing exists. In the contrary case, we report that no upward-planar 3-slope drawing exists.

Theorem 5.9. *If G is a plane directed inner triangulation on n vertices with maximum indegree and outdegree at most 3, then we can decide in $\mathcal{O}(n \log n)$ time whether G admits an upward-planar 3-slope drawing. Within the same time bound, we can construct such a drawing if it exists.*

Proof. We describe an $\mathcal{O}(n \log n)$-time algorithm that proceeds as follows. Take any triangular face of the input embedding as a start triangle and draw it such that the edge between the source and the sink has slope $\uparrow$ and length 2. For the other two edges take the slopes $\nwarrow$ and $\nearrow$ and length $\sqrt{2}$. Note that the given embedding prescribes which edge actually gets $\nwarrow$ and which edge gets $\nearrow$.

Now for each other triangular face f of the digraph where we have already drawn one of its edges, the exact shape of f follows by Observation 5.7. Hence, we can draw them all iteratively, e.g., by using a queue or a stack. As we have $\mathcal{O}(n)$ faces and edges, this procedure runs in $\mathcal{O}(n)$ time. If during the execution of this algorithm, there is a conflicting assignment of slopes, then there cannot be an upward-planar 3-slope drawing and we terminate.

So far, our algorithm finds an upward-planar 3-slope drawing of the upward-plane input digraph if it exists, however, it also may return false positive results. Namely, it remains to ensure that, even if we find a locally valid slope assignment for all edges, we do not have any intersecting or overlapping edges or vertices globally. (Imagine a path of triangles that can be drawn crossing-free locally, but intersects a non-neighboring face when wrapping around on the grid.) Now we exploit that all faces are drawn as triangles of the infinite triangular grid by Observation 5.8. This means that we cannot have edge–edge intersections or (inner) edge–vertex containments, but only vertex–vertex and edge–edge overlaps. In the latter case, we also have vertex–vertex overlaps.

Hence, we only need to assure that during the execution of the algorithm, no two vertices get assigned the same coordinate. Otherwise, there cannot be an upward-planar 3-slope drawing, and we terminate. Thus, we keep track of all coordinates we have already been assigned to vertices in a balanced binary search tree (the keys are the coordinates sorted lexicographically). This can be done in $\mathcal{O}(n \log n)$ time in total. $\square$

It follows that all subclasses of inner triangulations can be checked and drawn with three slopes by the same algorithm.

Corollary 5.10. *If G is a directed maximal outerpath with an upward-outerpath embedding or a maximal outerplane digraph on n vertices with maximum indegree and outdegree at most 3, then we can decide in $\mathcal{O}(n \log n)$ time whether G admits an upward-planar 3-slope drawing. Within the same time bound, we can construct such a drawing if it exists.*

It remains to consider the variable-embedding scenario. Given a digraph, we need to find a suitable inner-triangulation embedding for which we can then obtain an upward-planar 3-slope drawing by the algorithm from Theorem 5.9, or we need to make sure that no such embedding exists. Next, we describe a procedure how to obtain such an embedding.

Theorem 5.11. *Let G be a digraph on n vertices and with maximum indegree and outdegree at most 3. We can decide whether G is a directed inner triangulation, admits an upward-planar 3-slope drawing, and construct such a drawing if it exists in $\mathcal{O}(n \log n)$ time.*

Proof. By Observation 5.8, it is easy to see that in an upward-planar 3-slope drawing of an inner triangulation G, any 3-cycle in the underlying undirected graph $U(G)$ of G corresponds to a face in the drawing. In other words, all drawn 3-cycles are empty. Moreover, given a triangle $\langle u, v, w \rangle$ of $U(G)$, there is for each edge of that triangle, say uv, at most one vertex $x \notin \{u, v, w\}$ such that ux and vx are edges of $U(G)$.

Hence, we can incrementally construct an embedding of G. We find any 3-cycle C in G in $\mathcal{O}(n)$ time by considering all $\mathcal{O}(n)$ edges and checking for their endpoints all neighboring vertices, which is a constant number since we have maximum in- and outdegree at most 3. Now we have two possibilities to draw C; see Figure 5.12. We test both of them in separate runs. In one run, we then find, for each edge uv of C in constant time, a vertex $x \notin C$ with edges ux and vx if it exists (and abort if multiple of them exist). This gives us a new triangle $\langle u, v, x \rangle$, for which we continue recursively. This way, we construct our embedding in a BFS-like manner or report that no such embedding exists.

After we have found such an embedding, we apply the algorithm from Theorem 5.9. □

Again, this result also applies to subclasses of directed inner triangulations like maximal outerplanar digraphs. In the next sections, we will see, however, that having a few larger faces among many triangular faces suffices to make the problem NP-hard.

5.6 Outerplanar Digraphs

In this section, we show that for $k = 3$ slopes deciding whether an upward-outerplanar digraph admits an upward-outerplanar k-slope drawing is NP-hard. This result applies to both the fixed- and the variable-embedding scenario. This is in contrast to cactus digraphs (Section 5.4) and inner triangulations (Section 5.5), which also include maximal outerplanar digraphs. We will see that having a few inner faces bounded by four instead of three edges makes the problem NP-hard. Also, recall that for an arbitrary number of slopes, upward planarity for outerplanar digraphs can be decided in polynomial time as shown by Papakostas in 1994 [Pap94] – if an embedding is given, even in linear time.

We remark, however, that it remains open if the problem is also NP-complete. Containment in NP is not immediately clear since it is open whether some digraphs require irrational (or super-polynomial precise) coordinates for any k-slope drawing. More precisely, if the problem was in NP, there would be small proof certificates for yes-instances that a verifier could use to decide the problem in polynomial time. Typically a combinatorial characterization or a drawing of the input graph could act as such a certificate. However, in our case, we do not know whether there are digraphs that require irrational (or super-polynomial precise) coordinates and if so, how to treat them implicitly or, alternatively, how a combinatorial characterization would look like. There is a chance that the problem actually is $\exists\mathbb{R}$-complete, which we leave as an open question.

In the following, we describe our NP-hardness reduction for embedded outerplanar digraphs and 3 slopes. At the end of this section, we argue why this result is also applicable in the variable-embedding scenario. In the subsequent sections, we extend this NP-hardness construction to work for outerpaths and planar digraphs. From now on our drawings are rotated by 45° (upwards is to the top right) using the regular grid slope set $\{\uparrow, \nearrow, \rightarrow\}$. This rotation facilitates the visualization of large orthogonal structures in our figures.

5.6.1 Outerplane Digraphs and $k = 3$

We show that it is NP-hard to decide whether a given upward-outerplanar digraph with a given upward-outerplanar embedding admits an upward-planar drawing using only three distinct slopes by reduction from Planar Monotone 3-SAT.

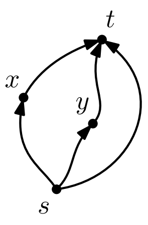
Planar Monotone 3-SAT (sometimes also known as Planar Monotone Rectilinear 3-SAT) is an NP-complete version of 3-SAT [dBK12], where the three literals of each clause are all either negated or unnegated – from now on called *negative* and *positive* clauses, respectively. In Section 3.3.2, we have already described an NP-hardness reduction from Monotone 3-SAT. In Planar Monotone 3-SAT, we additionally have that the incidence graph (defined in the next sentence) admits a planar drawing where the vertices are rectangles, the edges are vertical straight-line segments, the variables are arranged on a horizontal line, the positive clauses are above, and the negative clauses are below this line; see Figure 5.15a. The *incidence graph* of a SAT formula has a vertex for each variable and clause, and an edge for each occurrence of a variable in a clause between the corresponding vertices.

For our reduction, we can allow that a clause contains the same literal multiple times. Hence, we can also assume that all clauses contain exactly three literals (as otherwise, we can fill up clauses by duplicating literals and obtain an equivalent formula).

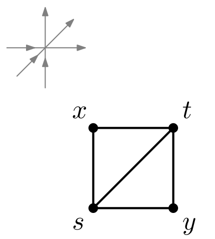
For a given Planar Monotone 3-SAT formula Φ and a rectangular drawing of its incidence graph, we construct a corresponding upward-outerplanar digraph G_Φ, which can only be drawn upward planar with 3 slopes if Φ is satisfiable. Our construction incorporates ideas of Nöllenburg [Nö05] and Kraus [Kra20], and heavily exploits the fact that some digraphs can only be drawn in a specific way when we restrict on upward-planar drawings with $k = 3$ slopes. In our reduction, we use them as building blocks of G_Φ. We describe these (sub)digraphs next.

Building Blocks. The most basic building block is the digraph $G_\square$; see Figure 5.14a. It consists of four vertices s, x, y, and t, and five edges st, su, ut, sv, and vt, which makes it two "glued" triangles; see also Section 5.5. Up to scaling and mirroring along a diagonal axis (essentially swapping vertices x and y), $G_\square$ admits an upward-planar 3-slope drawing only as a square as in Figure 5.14b. This fact follows immediately from Observation 5.7. Clearly, such a square is outerplanar. We can attach multiple squares (and triangles) to each other as in Figure 5.14c. The drawing of such a bigger digraph is unique up to scaling and mirroring along a diagonal axis, which follows from Observation 5.8. If the attached squares form a tree structure, the drawing is outerplanar. We refer to these squares as *unit squares*, since, once set, the side lengths for all attached squares are the same.

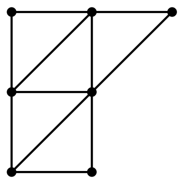
We also use $G_\square$ multiple times (from now on $G_\square$s for short) to construct our next building block – the digraph $G_\leftrightarrow$ as defined in Figure 5.14d. We consider $G_\leftrightarrow$ first without given embedding. We will then see that the upward-planar embedding is determined by the edge directions of the digraph. To allow a certain small degree of freedom in our construction, we exploit the properties of $G_\leftrightarrow$ that are specified in Lemma 5.12.



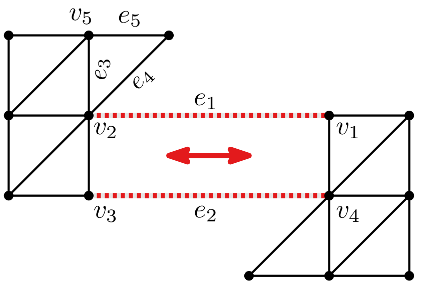
(a) digraph $G_\square$



(b) unique 3-slope realization of $G_\square$ up to swapping x and y



(c) combination of $G_\square$s and triangles



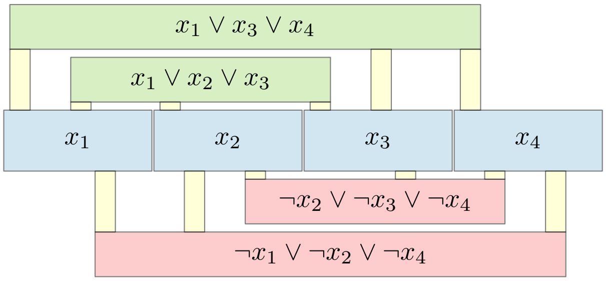
(d) digraph $G_\leftrightarrow$

Figure 5.14: Building blocks used as sub-digraphs in our NP-hardness reduction. The digraph $G_\square$ admits only an upward 3-slope drawing as a square. By combining copies of $G_\square$ and triangles, we can build larger rigid structures. Then, all $G_\square$s have the same size and we refer to them as unit squares. The digraph $G_\leftrightarrow$ admits only upward 3-slope drawings with one degree of freedom.

Lemma 5.12. *In any upward-planar 3-slope drawing of $G_\leftrightarrow$ (see Figure 5.14d)*

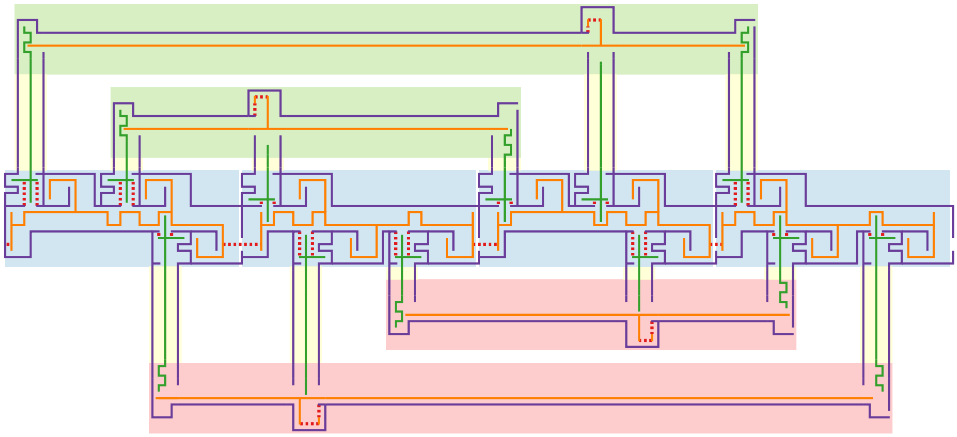
- *the edges e_1 and e_2 are parallel and have the same arbitrary length $\ell > 0$,*
- *all edges are oriented as in Figure 5.14d up to mirroring along a diagonal axis, and*
- *all vertical and horizontal edges (excluding e_1 and e_2) have the same length. Also all diagonal edges have the same length.*

Proof. As subgraphs, we have twice a pair of attached $G_\square$s. As observed before, each pair can only be drawn as attached unit squares. Without loss of generality, let the first two squares (containing vertices v_2, v_3, v_5) be drawn vertically above each other as in Figure 5.14d. Since v_2 has outdegree 3 and $e_3 = v_2v_5$ has slope $\uparrow$, we can argue that the edge e_1 gets slope $\rightarrow$. Indeed, if e_1 used $\nearrow$, the edge e_5 would have no slope to close the triangle bounded by $\langle e_3, e_4, e_5\rangle$ because then, at v_5, the only remaining slope for e_5 would be $\rightarrow$. Thus, e_4 gets $\nearrow$ and the only remaining slope for e_1 is $\rightarrow$. Note that e_3 is a unit square edge and the adjacent edges e_4 and e_5 form a triangle



$x_1 \lor x_3 \lor x_4$
$x_1 \lor x_2 \lor x_3$
x_1
x_2
x_3
x_4
$\neg x_2 \lor \neg x_3 \lor \neg x_4$
$\neg x_1 \lor \neg x_2 \lor \neg x_4$

(a) Rectangular incidence graph drawing of a Planar Monotone 3-SAT formula Φ.



(b) Outerplanar drawing of the digraph G_Φ. Chains of unit squares are drawn as straight-line segments. The variable/clause/edge gadgets occupy the areas of their corresponding rectangles. Here, x_1 and x_4 are set to false (brush on the left side within their variable gadgets), while x_2 and x_3 are set to true (brush on the right side within their variable gadgets)

Figure 5.15: Schematic example for our NP-hardness reduction.

with e_3 that has the same size as half a unit square. Hence, e_4 has the same length as all diagonal edges and e_5 has the same length as all horizontal and vertical edges.

The same argument as for the outgoing edges of v_2 applies to the incoming edges of v_4. However, assume for a contradiction, that the whole block on the right side of Figure 5.14d is mirrored along a diagonal axis, i.e., v_1 and v_4 are on the same horizontal line and e_2 gets slope $\uparrow$. Then, v_4, v_1, and v_2 would be on the same horizontal line (the line hosting e_1). This contradicts v_3 being on the same vertical line as its neighbors v_2 and v_4. Hence, the right block has the same orientation as the left one, and e_2 gets slope $\rightarrow$ and is parallel to e_1. This implies that the distances between v_2 and v_3 on the one hand, and v_1 and v_4 on the other hand are the same and the $G_\square$s (and their attached triangles) have the same size and the orientation as in Figure 5.14d. By the orientations of v_1, v_2, v_3, v_4, the edges e_1 and e_2 have identical length ℓ. Only a value $\ell \leq 0$ would cause a non-planar drawing. $\square$

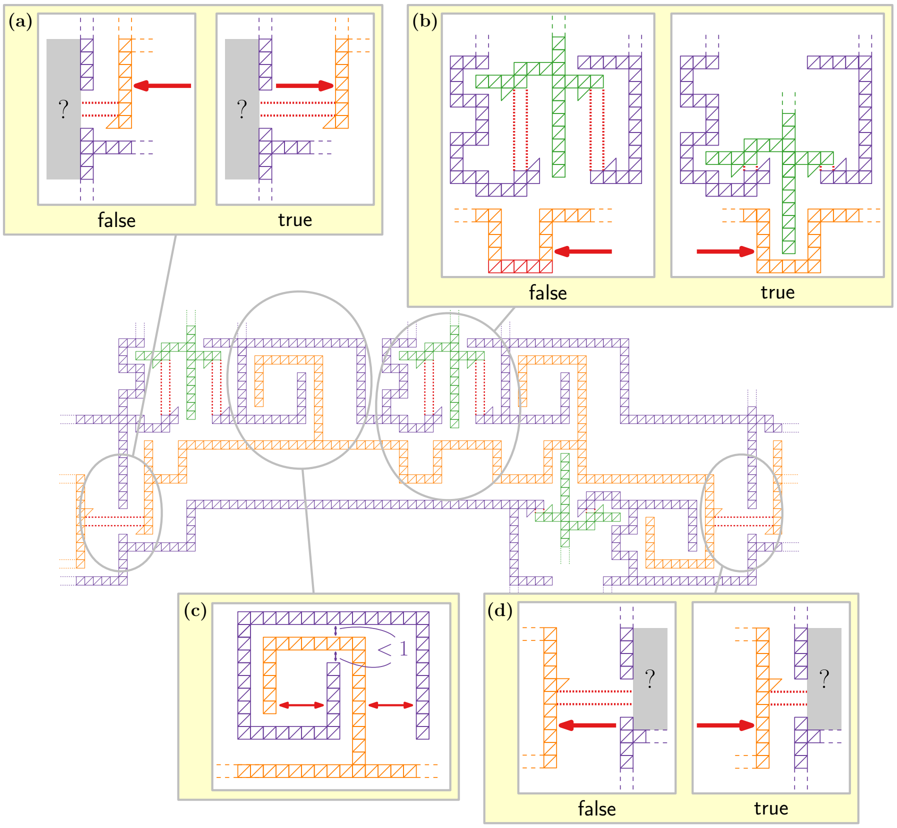


Figure 5.16: A variable gadget, which is contained in two positive and one negative clauses. The brush is positioned to the left and, thus, the variable is set to false.

With this construction kit of useful (sub)digraphs in hand, we build a digraph whose upward-planar drawings represent the satisfying truth assignments of Φ.

Overview. The high-level construction is depicted in Figure 5.15b. We construct, for each variable, a specific digraph – a *variable gadget* (blue in Figure 5.15b). Similarly, for each clause, there is a specific digraph – a *clause gadget* (green and red in Figure 5.15b). All gadgets mainly consist of chains of $G_\square$s. In an upward-planar drawing, this enforces a rigid frame structure built from unit squares. We "glue" all variable gadgets together in a row and connect variable and clause gadgets by *edge gadgets* (yellow in Figure 5.15b) such that the composite digraph remains upward outerplanar and all $G_\square$s are drawn as unit squares.

Variable Gadget. A variable gadget is depicted in Figure 5.16. Its base structure is the (violet) frame composed of chains of unit squares. The core element is the (orange) central chain of unit squares (with a few side-arms), which has one degree of flexibility, namely, moving as a whole to the left or to the right without leaving the frame structure of the gadget. It looks and behaves a bit like a pipe cleaning brush that is stuck inside the frame but can be moved a bit back and forth. Hence, we call it a *brush*. It is connected via a $G_{\leftrightarrow}$ to the brush of the previous variable gadget (see Figure 5.16a/d) and the first brush is connected to the frame via a $G_{\leftrightarrow}$ (see on the left side of Figure 5.15b). This allows only a horizontal shift of the brushes, but no vertical movement relative to its anchor point at the frame structure. Note that the horizontal position of the brush in any variable gadget is independent of those in all other gadgets. If the brush is positioned to the very left (right), the corresponding variable is set to false (true).

For each occurrence of a variable in a positive clause, we have a construction as depicted in Figure 5.16b. There, a long chain of (green) $G_{\square}$s – from now on called *bolt* – is attached to the frame structure via two $G_{\leftrightarrow}$s, which allow only a vertical, but no horizontal shift. The bolt has on its left side an arm, which can only be placed in one of two pockets of the frame. It can always be placed in the upper pocket, which pushes the bolt outwards with respect to the variable gadget (it is pushed deeper into an edge gadget and a clause gadget). It can only be placed in the lower pocket if the brush is shifted to the very right (i.e., set to true) – then the bolt can “fall” into a cove of the brush. For each occurrence of a variable in a negative clause, we have this construction upside-down and such that the bolt can be pulled into the variable gadget only if the brush is shifted to the very left (i.e., set to false).

Note that, to maintain outerplanarity of the whole construction, the frame structure is not contiguous, but connected by $G_{\leftrightarrow}$s and the arms of the bolts. Hence, the frame structure decomposes into many components that have fixed relative horizontal positions and their unit squares have the same side lengths. However, the components can shift up and down relative to each other. To keep this vertical shift small enough not to affect the correct functioning of our reduction, we use, for each such component, the construction depicted in Figure 5.16c. The chain of brushes has no vertical flexibility and serves as a base ground for “anchors” of the frame. The frame can move less than one unit up or down unless it violates planarity. If the frame would be shifted up enough to be completely above the brush, the previous part of the frame would intersect the arms of the adjacent bolt.

Edge Gadget. An edge gadget consists of only three straight chains – two frame segments and a bolt in the middle. Their purpose is to fix the distance between a clause gadget and a variable gadget and to synchronize the size of the unit squares between the variable and clause gadgets. Several edge gadgets are depicted on yellow background color in Figure 5.15b.

Clause Gadget. A clause gadget for a positive clause is depicted in Figure 5.17. Within a frame, which is connected at six points to the frames of three edge gadgets, there is a horizontal (orange) bar, which is attached via two $G_{\leftrightarrow}$s to the frame – one $G_{\leftrightarrow}$ allows a horizontal, the other allows a vertical shift. It resembles a crane that can move up and extend its arm, while it holds the horizontal bar on a vertical (orange) rope. The three bolts from the corresponding variable gadgets reach into the clause

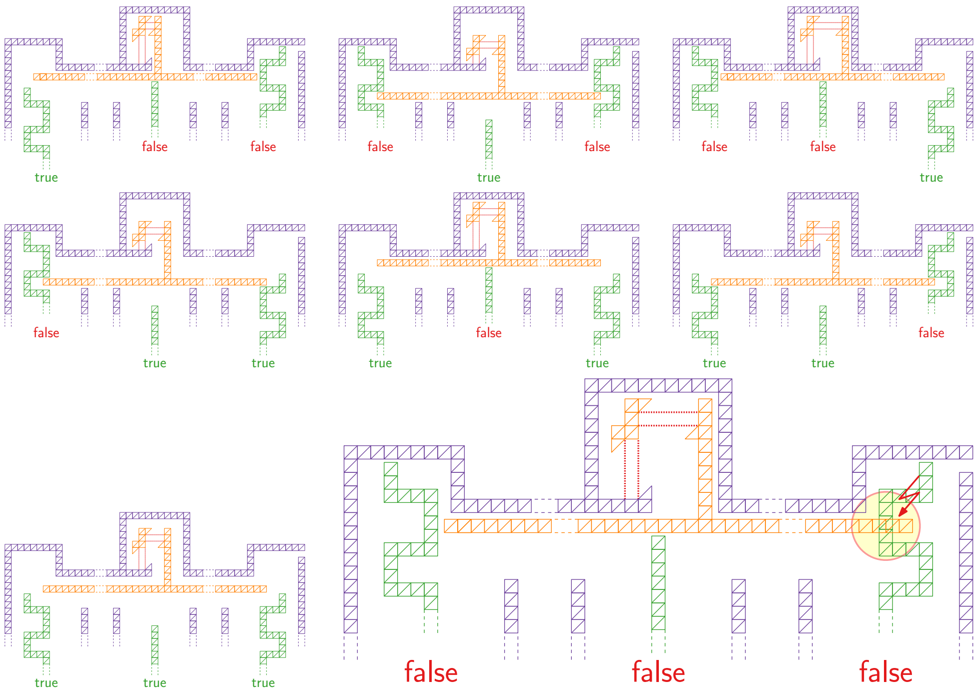


Figure 5.17: Positive clause gadget in eight configurations.

gadget. The lengths of these bolts is chosen such that, if they are pushed out of their variable gadget and into the clause gadget, they only slightly fit inside the gadget. Depending on whether each of the bolts is pushed into the clause gadget or pulled out of it, we have eight possible configurations (with sufficiently small vertical slack). They represent the eight possible truth assignments of a clause. In Figure 5.17, we illustrate that in each configuration, we can accommodate the horizontal bar in an upward-planar 3-slope drawing of the clause gadget – except for the case when all three bolts push into the clause gadget, which represents the truth assignment false to all contained variables.

A negative clause gadget uses the same construction but mirrored vertically. There, three bolts pushing into the clause gadget means that the contained variables are all set to true.

Putting our gadgets together, we conclude:

Theorem 5.13. *Deciding whether an upward-outerplane digraph admits an upward-planar 3-slope drawing is* NP*-hard.*

Proof. To show NP-hardness, we use the reduction from Planar Monotone 3-SAT described above. Let Φ be a given Planar Monotone 3-SAT formula with a rectangular drawing of the incidence graph of Φ, and let G_Φ be the digraph obtained by our reduction. Each gadget in G_Φ has only polynomial size, and we can construct it in polynomial time.

If Φ is satisfiable, then there is a satisfying assignment $T^\star$ of truth values to the variables of Φ. Draw G_Φ as illustrated in Figures 5.15b, 5.16 and 5.17 such that the brush in a variable gadget has a distance of ε (for some sufficiently small $\varepsilon > 0$) to the frame on its left if the corresponding variable is set to false in $T^\star$ and has a distance of ε to the frame on its right if the corresponding variable is set to true. This drawing uses only three slopes and is upward planar. Observe that it is also outerplanar. On a chain of unit squares, the vertices on both sides are incident to the outer face. The drawings of the variable gadgets are open from one to the other and the drawing of the rightmost variable gadget is open to the outside. Moreover, the drawings of the positive (negative) clause gadgets are open on their top (bottom) right and on their whole bottom (top) sides, which also gives the "nested" clause gadgets and the outer sides of the variable gadgets access to the outer face.

On the other hand, if there is an upward-planar 3-slope drawing of G_Φ and it resembles the structure in Figure 5.15b, we can read off a satisfying truth assignment depending on the positions of the brushes (for the ones in intermediate position where no bolt can use the lower pocket, we can use any assignment for this variable).

In the remainder of this proof, we argue that any drawing of the embedded G_Φ resembles this structure. Consider the $G_\square$ of the frame structure which is connected via a $G_\leftrightarrow$ to the brush of the first variable gadget. Clearly, it is drawn as a unit square – it is our reference unit square having side length 1. Observe that all other unit squares are connected via chains of $G_\square$s or $G_\leftrightarrow$s to this reference unit square. By our observations and Lemma 5.12, they have the same side length. When ignoring the central parallel edges e_1 and e_2 in the $G_\leftrightarrow$s, the drawing decomposes to few rigid components. The first rigid frame component contains the reference unit square. The central edges of the incident $G_\leftrightarrow$ connecting it to the first brush can have only relatively small length as they would hit the "back wall" of, again, the first rigid frame component otherwise. Hence, the brush is indeed drawn inside the frame of the first variable gadget. Moreover, the first rigid frame component is connected via a $G_\leftrightarrow$ to a bolt, which in turn is connected to the next rigid frame components of the first variable gadget (see Figure 5.16b). Since this bolt cannot escape the upper boundary of the clause gadget, which is also part of the first rigid frame component, the arm is in one of the two pockets. Then, the start point of the next rigid frame structure cannot be above the arm of the brush in Figure 5.16c and the construction depicted there keeps the vertical slack of this rigid frame component within the range $(-1, 1)$. This argument inductively propagates for all following rigid components. Hence, we have vertical slack of less than 1 for the frame structures and for the arms in the pockets. In the configuration false-false-false (true-true-true) of a positive (negative) clause gadget, this would give us $2 - \varepsilon$ vertical slack if two frames move away from each other and additionally $1 - \varepsilon$ vertical slack if the arm is close to the bottom of its pocket. However, we would require a vertical shift of the horizontal bar of more than 3 to be below or above the bulge of the left or the right bolt within a clause gadget. Hence, this configuration is not drawable. $\square$

5.6.2 Outerplanar Digraphs and $k = 3$

Note that in G_Φ, we have used only connected $G_\square$s and $G_\leftrightarrow$s. By the rigidity of chains of unit squares (see also Observation 5.8 and Theorem 5.11) and by Lemma 5.12, the planar embedding of G_Φ is unique up to mirroring along a diagonal axis. Therefore, our reduction holds true also for the variable-embedding scenario and we derive the following corollary from Theorem 5.13.

Corollary 5.14. *Deciding whether an upward-outerplanar digraph admits an upward-planar 3-slope drawing is* NP*-hard in the variable-embedding scenario.*

5.7 Outerpaths

The subclass of outerplanar digraphs whose weak dual graph is a path, are called (directed) outerpaths. Next, we show that we can modify the NP-hardness reduction of Section 5.6 to work also for outerpaths. As for the outerplanar digraphs, this result applies only to three slopes and we show only NP-hardness, but not containment in NP. Restricting the graph class to outerpaths makes it a strictly stronger result. However, we have started with the reduction for outerplanar digraphs because it is conceptually easier and can be extended to more than three slopes for planar digraphs. Upward-planar 3-slope drawings of directed outerpaths are also the topic of Geis' bachelor thesis [Gei22].

Theorem 5.15. *Deciding whether a directed outerpath with an upward-outerpath embedding admits an upward-planar 3-slope drawing is* NP*-hard.*

Proof. We modify the NP-hardness reduction from Section 5.6.1, where we have created an outerplane digraph G_Φ from a Planar Monotone 3-SAT formula Φ. We name the new embedded outerpath and its intermediate steps G'_Φ. The high-level idea to build G'_Φ is to walk along the outer face of the embedded G_Φ and to replace this boundary by a long chain of (more) unit squares. The weak dual of G'_Φ is then a cycle around a central vertex[12] corresponding to a large slim inner face f_{in} that represents the interior of G_Φ. If we remove one unit square of G'_Φ, such that the outer face and f_{in} merge, then the weak dual is a path and G'_Φ becomes an outerpath.

We replace the tree-structure of unit squares in G_Φ by chains of more unit squares as illustrated in Figure 5.18. If we were to superimpose the two drawings, the ratio of the side lengths of the unit squares would be $1 : 16$. However, we have to be a bit careful in some of the corners to maintain the outerpath property. There, we use triangles instead of unit squares; see the yellow boxes in Figure 5.18. Triangles also preserve the unit size as observed in Observation 5.8.

It remains to replace all $G_\leftrightarrow$s. Here, the replacement is more intricate. We call the replacement $G^+_\leftrightarrow$ and it is depicted in Figure 5.19. Instead of the two edges e_1 and e_2 of equal but arbitrary length in $G_\leftrightarrow$, we have two slightly reduced $G_\leftrightarrow$s inside $G^+_\leftrightarrow$. Observe that the top left and the bottom right triangle is missing, which however

[12] This graph is known as the *wheele graph*, which we encounter also in Section 5.8.2.

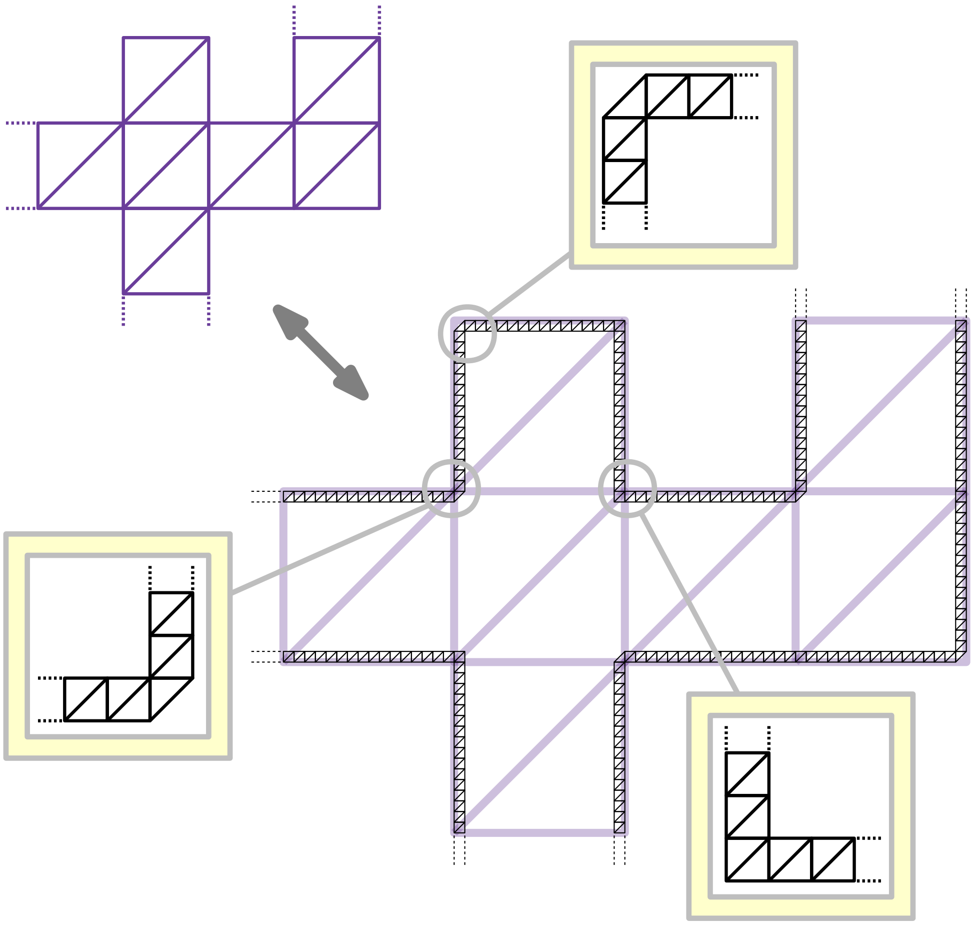


Figure 5.18: We replace the outer boundary of the unit squares of our NP-hardness reduction for outerplanar digraphs by long chains of "smaller" unit squares resulting in an outerpath.

does not affect the functionality of $G_{\leftrightarrow}$ and the properties described in Lemma 5.12. Consequently, the properties of Lemma 5.12 hold for both of them, however, we need to assure that they are synchronized. In Figure 5.19, consider the chains of unit squares attached to both $G_{\leftrightarrow}$s. They allow a vertical and horizontal slack of at most ± 1 unit with respect to the new (smaller) unit squares. However, since the flexible parts are only the two horizontal $G_{\leftrightarrow}$s, the offset originating inside $G^{+}_{\leftrightarrow}$ is ± 1 in horizontal, but nothing in vertical direction. We still have to be a bit more careful because one can "unhinge" the two parts of $G^{+}_{\leftrightarrow}$ by a vertical shift of $24 + \varepsilon$ for an $\varepsilon > 0$, which is less than two original units.

Therefore, we also have to reduce the slack of the base construction. We tighten the two pockets that we have in a variable gadget for each occurrence of that variable in a clause (see Figure 5.16b) to only ± 1 new units of vertical slack. Also, we extend

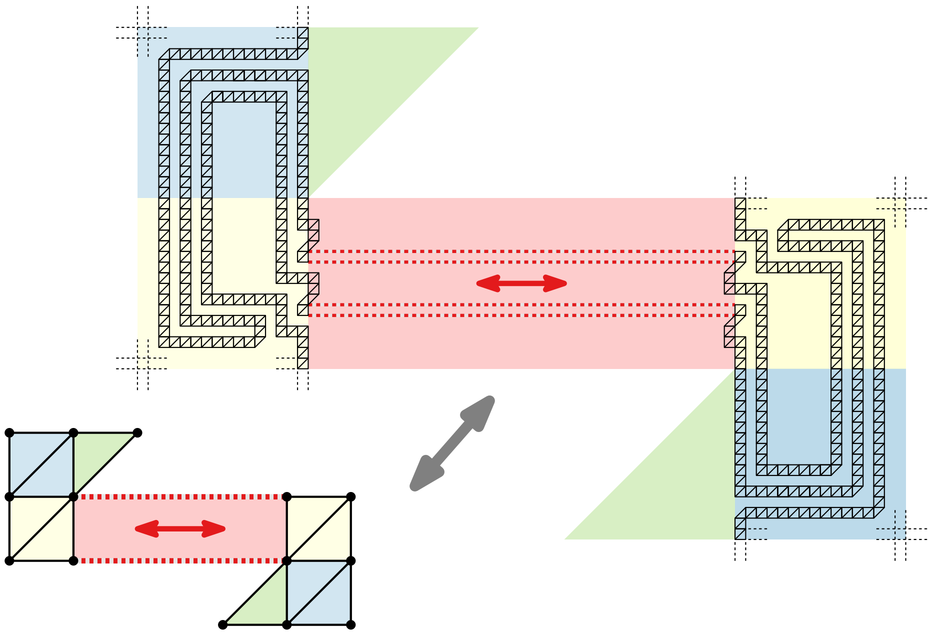


Figure 5.19: Transformation of $G_{\leftrightarrow}$ (bottom left) to parts of an outerpath, which we call $G^{+}_{\leftrightarrow}$ (top right). The two edges of variable length are replaced by two (slightly reduced) $G_{\leftrightarrow}$s.

the arm of the bolt to have a horizontal distance of 1 new unit to the frame. Similarly, we proceed for the hook-like structure in Figure 5.16c. We extend the arm of the brush such that the vertical slack is ± 1 new units. This assures that the frame structure admits an upward-planar 3-slope drawing only in the illustrated way with vertical slack of ± 1 since the $G^{+}_{\leftrightarrow}$s connecting the brush do not allow vertical slack; see Theorem 5.13.

It remains to consider the clause gadgets. The bar and the rope are all connected by unit squares, so they allow no slack for the horizontal $G^{+}_{\leftrightarrow}$ in a clause gadget. The vertical $G^{+}_{\leftrightarrow}$ receives ± 1 horizontal slack by the horizontal $G^{+}_{\leftrightarrow}$, which is not sufficient to “unhinge” the two parts of $G^{+}_{\leftrightarrow}$. In Theorem 5.13, we have argued that we would need a vertical shift of more than 3 old units to break the clause gadget, but the vertical shift is no more than $3 - \varepsilon$ old units due to the pockets and the arms of the brush. We have adjusted the latter ones to admit also no more than $3 - \varepsilon$ new units vertical slack. Then, even though we have additionally ± 2 vertical slack due to the vertical $G^{+}_{\leftrightarrow}$ of a variable gadget and the vertical $G^{+}_{\leftrightarrow}$ of a clause gadget, this does not exceed $3 \cdot 16 = 48$ new units and the proof of Theorem 5.13 remains valid. $\square$

Clearly, due to Theorem 5.11 and Lemma 5.12, the embedding is unique up to mirroring. Hence, we conclude the following corollary.

Corollary 5.16. *Deciding whether a directed outerpath admits an upward-planar 3-slope drawing is* NP*-hard in the variable-embedding scenario.*

5.8 Planar Digraphs

In this section, we show that for any constant $k \geq 3$ deciding whether an upward-planar digraph admits an upward-planar k-slope drawing is NP-hard. Except for $k = 4$, this hardness holds true regardless of whether we prescribe an embedding or not. We obtain these results by adjusting the NP-hardness reduction of Section 5.6. We will see that we can, by a relatively simple extension, allow more than three slopes at the price of giving up outerplanarity. Again, containment in NP remains an open question.

5.8.1 Plane Digraphs and $k \geq 4$

Consider the embedded digraph G_Φ. We can add dummy leaves to each vertex to occupy all but the originally used slopes. Since any 3 slopes can be projected to $\{\uparrow, \nearrow, \rightarrow\}$ and we block all other slopes, our arguments work for all sets of k slopes and the reduction remains correct. The digraph remains upward planar, but it is not upward-outerplanar anymore. We formalize this statement in the following corollary derived from Theorem 5.13.

Corollary 5.17. *Deciding whether an upward-plane digraph admits an upward-planar drawing with k slopes is NP-hard for $k \geq 3$. This holds true for all choices of k slopes.*

5.8.2 Planar Digraphs and $k \geq 5$

Last, we consider the variable-embedding scenario. We remark that, given a digraph, finding an upward-planar embedding is already NP-hard [GT01] for all $k \geq 2$ [KM22]. This NP-hardness immediately propagates to our problem. However, we can show that the problem remains NP-hard even if we can find an embedding (or multiple different embeddings) efficiently. Hence, NP-hardness additionally comes from finding a concrete drawing when we are allowed to change any upward-planar embedding arbitrarily. In other words, even if an oracle tells us the upward-planar embeddings of a given digraph G, deciding whether G admits an upward-planar k-slope drawing for $k \geq 5$ slopes remains NP-hard.

We show that our NP-hardness reduction remains applicable for $k \geq 5$ slopes by extending our digraph G_Φ. This leaves $k = 4$ in the variable-embedding scenario as the only open case. More precisely, we extend G_Φ such that it has a unique planar embedding up to mirroring along a diagonal axis and up to swapping subgraphs only used to occupy the slopes we do not use for our original NP-hardness reduction with three slopes.

Assume for now that k is an odd number; later, we consider the case that k is even. From the given k slopes, we pick the three middle slopes to host the digraph of the hardness construction described before. For simplicity, we visualize these three middle slopes again as $\{\uparrow, \nearrow, \rightarrow\}$ and the other slopes in quadrants II and IV, i.e., the quadrants on the top left and the bottom right, around a vertex. The key idea is to occupy the unused slopes at each vertex by *fans* and *beaters* as depicted in Figures 5.20a and 5.20b instead of simple leaves.

Fans are appended to the outer face at a vertex v if the angle α that has been formed in the old construction is at least $180°$. More precisely, for each unoccupied slope spanned by α, we add a neighbor to v and then connect each consecutive pair of these new neighbors of v; edge directions are set appropriately. For each other remaining slope s at each vertex v, we add a beater.

A beater is a digraph obtained from the wheel graph W_{2k+1} as follows. The *wheel graph* W_{2k+1} is the cycle $C_{2k} = \langle v_1, \dots, v_{2k} \rangle$ with an additional vertex c that is adjacent to all other vertices $\{v_1, \dots, v_{2k}\}$. In our upward-planar setting, we now consider a directed version of W_{2k+1}. The edges of W_{2k+1} on the cycle C_{2k} are directed from a local source $v_{\lfloor k/2 \rfloor}$ to a local sink $v_{k+\lfloor k/2 \rfloor}$, the vertices $v_1, \dots, v_k$ have outgoing edges towards c, and the vertices $v_{k+1}, \dots, v_{2k}$ have incoming edges from c. We call the edges incident to c *spokes*. For a particular $i \in \{1, \dots, 2k\}$, we remove the vertex v_i and replace the spoke cv_i by the spoke $e^\star = cv$ (the edge direction may be inverse depending on i); see Figure 5.20b. This construction enforces an order on the spokes and, hence, we choose i such that we can prescribe the slope s of $e^\star$. Note that the whole beater could be mirrored along a diagonal axis leaving two possible slopes for $e^\star$. However, this is unproblematic since in our construction the "mirrored" slope of s is also occupied by a beater or a fan. For an illustration of how to append fans and beater see Figure 5.20c.

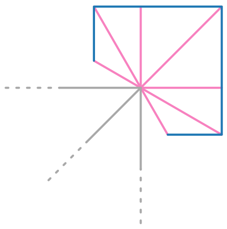
Next, we prove that this suffices to enforce an embedding of the prescribed structure and we describe how to extend the construction when k is an even number. Though we lost upward outerplanarity, note that the underlying undirected graph remains outerplanar for odd k.

Theorem 5.18. *Deciding whether an upward-planar digraph admits an upward-planar drawing with k slopes is* NP*-hard for $k \in \mathbb{N}^+ \setminus \{1, 2, 4\}$ in the variable-embedding scenario even if we can find an upward-planar embedding efficiently. This holds true for all choices of k slopes.*

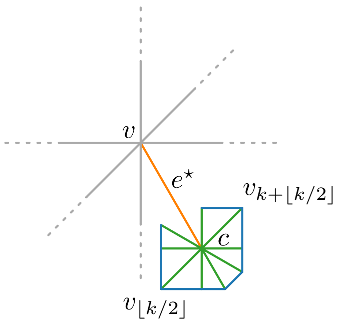
Proof. For $k = 3$, we use Corollary 5.14. Now assume that $k \geq 5$ is an odd number. We consider the case that k is even afterwards. We show next that the extended digraph can only be embedded in the previously described way up to mirroring along a diagonal axis and up to swapping beaters at a vertex.

Observe that the (pink) edges of each fan occupy neighboring slopes because their other endpoints are connected by (blue) edges. For every fan, the slope set it covers contains both incoming and outgoing edges. Therefore, there are at most two ways to add a fan to a vertex – either to its left or to its right side. In any case, it needs to be added to the outer face since otherwise neither a $G_\square$ nor a $G_\leftrightarrow$ can be drawn (any fan blocks $\geq k-1$ slopes, which corresponds to $\geq 180°$).

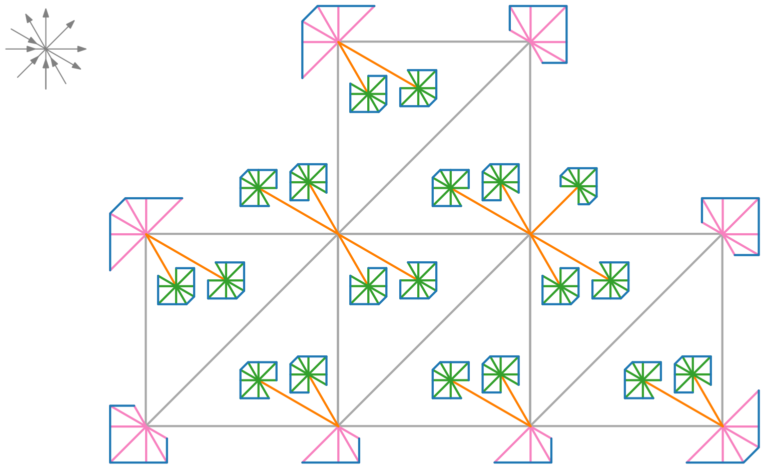
The inner (green) edges of a beater occupy all but one slope, which remains for the (orange) edge $e^\star$ connecting the beater to its vertex. We next analyze the slope of $e^\star$. Because of the outer edges of a beater, there is an order of the slopes of the inner edges (up to mirroring the whole beater) which determines the assignment of slopes within the beater. Hence, edge $e^\star$ uses one of two possible slopes – say



(a) A fan.



(b) A beater.



(c) We add fans and beaters to each vertex of the digraph such that all unused slopes are occupied.

Figure 5.20: Example for $k = 5$ illustrating how to occupy unused slopes such that our NP-hardness reduction with three slopes remains applicable.

the ℓ-th or the $(k - \ell + 1)$-th slope. Since k is an odd number, these slopes are identical in case ℓ is the middle slope $\lceil k/2 \rceil$, namely $\ell = \lceil k/2 \rceil = k - \ell + 1$, which fixes the position of some beaters (e.g., in Figure 5.20c, the beater in the outer face on the right side). For the other beaters, observe that we use the $(\lceil k/2 \rceil - 1)$-th and the $(\lceil k/2 \rceil + 1)$-th slope never for a beater, but always for the edges of the original construction or fans. For all other values of ℓ, the opposite $(k - \ell + 1)$-th slope is always occupied by another beater or a fan.

This means that the embedding of the subgraph used for the reduction is the same as in the digraph G_Φ from the reduction of Theorem 5.13 up to mirroring, and the embedding of the whole digraph is unique up to mirroring and exchanging the positions of pairs of beaters.

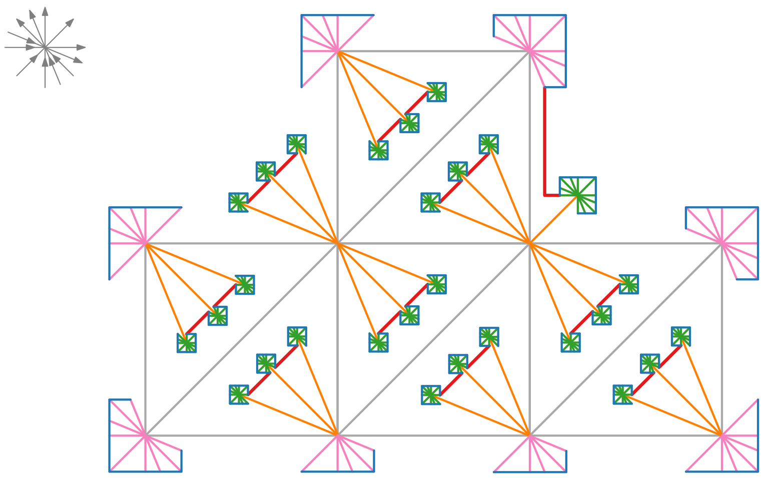


Figure 5.21: Example for $k = 6$ illustrating how to occupy unused slopes such that our NP-hardness reduction with three slopes remains applicable. We add (red) dummy edges to connect the beaters to each other or an adjacent fan, which fixes the faces, in which the beaters end up, in the case k is even and greater than 4.

It remains to consider the case when $k \geq 5$ is even. Our goal is to use for the digraph from Theorem 5.13 always the same three middle slopes, e.g., the $(k/2-1)$-th, $k/2$-th, and $(k/2+1)$-th slope as in Figure 5.21. This is automatically the case if all fans and beaters are drawn in their designated faces. We next describe how we can enforce in which face a beater or fan is drawn. At each vertex, we connect all incident beaters that are supposed to share a common face by (thick red) dummy edges as depicted in Figure 5.21. If there is only one beater at a vertex at a face, we connect it by a path of length 2 to a neighboring fan; see the one on the top right of Figure 5.21. This latter case concerns only beaters on the outer face that block the slope $\nearrow$. (Inside a $G_{\leftrightarrow}$, we do not need beaters between e_1 and e_2.) In every other case, observe that within each bundle of connected beaters being adjacent to a vertex v, there is at least one beater being connected to v by an outgoing edge, and there is at least one beater being connected to v by an incoming edge. Hence, this bundle must be placed in the face being incident to v where the direction of (gray) bounding edges switches. This is either the opposite side of a fan or the opposite side of another bundle of beaters. In the latter case, the bundle of beaters cannot be exchanged since this would leave an incoming and an outgoing slope unused, which is not possible if v has in- and outdegree k. □

5.8.3 Planar Digraphs and $k = 4$

We have shown that it is NP-hard to decide whether a given digraph admits an upward-planar drawing with k slopes for all $k \geq 3$ in the fixed- and the variable-embedding scenario except for $k = 4$ in the variable-embedding scenario. The reason that we could not show NP-hardness for this specific setting is due to our extended hardness construction using beaters and fans. We use beaters and fans to block all but three designated slopes for the base hardness construction on three slopes. However, beaters can be drawn in two different ways (by mirroring) and hence potentially block two different slopes. The slopes blocked by the same beaters can be grouped into pairs. If we have an odd number of slopes, the central slope is not part of such a pair (a mirrored beater still blocks the central slope). We thus use the central slope and a pair of slopes for our three slopes in the base construction.

If k is even, we connect beaters in order to block a sequence of slopes at a vertex v – these are slopes for both incoming and outgoing edges incident to v. We exploit the property that at v, the incoming and outgoing edges form a contiguous sequence in any upward embedding and, hence, there are two turning points of edge directions at v. Choosing three neighboring slopes from an even number of available slopes leaves two sets of unused neighboring slopes (then blocked by beaters) whose cardinalities are off by one. Hence, at a vertex with degree $2k$, the number of incident incoming and outgoing edges belonging to a group of connected beaters differs for the left and the right turning point. For $k = 4$ we use all but one slope for the base construction. This means that we cannot connect beaters and, thus, cannot cover these turning points at the vertices. Hence, a beater cannot reliably block the fourth slope but may use one of the three slopes of the base construction and an edge of the base construction may then use the fourth slope.

5.9 Concluding Remarks and Open Problems

We have investigated the problem of whether a digraph admits an upward-planar k-slope drawing for a fixed set of k slopes. Roughly speaking, for three slopes, the boundary between polynomial-time solvability and NP-hardness lies between inner triangulations and almost-maximal outerpaths having a linear number of degree-4 faces. For constant $k > 3$ slopes, this boundary lies somewhere between cacti and planar digraphs. In our analysis, we have somewhat sidestepped the issue of representing the coordinates of our drawings. This is also the reason why we could show only NP-hardness rather than NP-completeness for our problems. Like for some other geometric problems, there is a chance that our problem is $\exists\mathbb{R}$-complete. (Remember that determining the (upward) planar slope number is $\exists\mathbb{R}$-hard in general [Hof17, Qua21].)

Further open questions remain. Let us start with the question marks in Table 5.1. For planar digraphs, the only open case is $k = 4$ in the variable-embedding scenario; in Section 5.8.3, we discuss why our current hardness construction fails here. For outerplanar digraphs, directed outerpaths, and directed inner triangulations, it is unclear whether deciding if there exists an upward outerplanar k-slope drawing is NP-hard for $k > 3$. Supposed these cases were NP-hard, where does this decision problem become polynomial-time solvable? Containment in P is not even clear for cacti and $k \in \omega(1)$, where we could give only an FPT-algorithm. One may try to find an optimization to our dynamic program to get rid of the dependence on k in the exponent in the running time of the algorithm. Furthermore, one may extend the algorithm to the regular-grid setting or to an arbitrary set of k slopes.

In general, area consumption and area requirement is a question worth investigating for many of our problem variants. In particular, it remains open whether an upward-planar k-slope drawing of an unordered directed tree with maximum in- and outdegree k sometimes requires exponential area.

The slope number is also related to the segment number, which we have investigated in Chapter 4. As far as we know, the segment number of upward-planar drawings has not been studied yet. It would be interesting to understand the connection between these numbers.

Chapter 6
Coloring Mixed and Directional Interval Graphs

In the last chapter of the theoretical graph drawing part, we provide the theoretical foundation for one step of our algorithm described in the applied graph drawing part. There, we investigate drawings of technical networks like cable plans. Commonly, technical networks are drawn with straight-line segments in the *orthogonal* style, i.e., all (or almost all) segments are axis-parallel, i.e., horizontal or vertical. Moreover, the components of technical networks are often drawn on few discrete layers. Between these layers, we aim for a compact orthogonal drawing of the edges. Such compact drawings are an application for directional interval graphs – specific mixed graphs, which we consider next.

6.1 Introduction

Mixed graphs are a powerful tool to model a network with different types of relations – directed and undirected relations. It has also been closely connected with the task of graph coloring. Classical *(vertex) coloring* of an undirected graph is an assignment of positive integers (called *colors*) to vertices such that every two neighboring vertices have distinct colors. The objective is to minimize the maximum color. For a graph G, the chromatic number $\chi(G)$ is the smallest number of colors in any coloring of G.

For mixed graphs, this concept generalizes in different ways naturally. We use the following generalization, which we call *proper coloring*. A *proper coloring* of a mixed graph G is a function c that assigns a positive integer to each vertex in G, satisfying $c(u) \neq c(v)$ for every edge $\{u, v\}$ in G, and $c(u) < c(v)$ for every arc (u, v) in G. It is easy to see that a mixed graph admits a proper coloring if and only if the arcs of G do not induce a directed circuit. For a mixed graph G with no directed circuit, we define the chromatic number $\chi(G)$ as the smallest number of colors in any proper coloring of G. Otherwise we let $\chi(G) = \infty$.

Previous Work. The concept of mixed graphs was introduced by Sotskov and Tanaev in 1976 [ST76] and it was reintroduced by Hansen, Kuplinsky, and de Werra [HKdW97] in the context of proper colorings of mixed graphs.

Coloring of mixed graphs was used to model problems in scheduling [FKŻ08b, KHOO17]; see also a survey by Sotskov [Sot20]. Without restrictions on the graph class, the problem is NP-hard because it directly generalizes the classical NP-hard problem of (undirected) graph coloring [Kar72]. However, it was also considered for

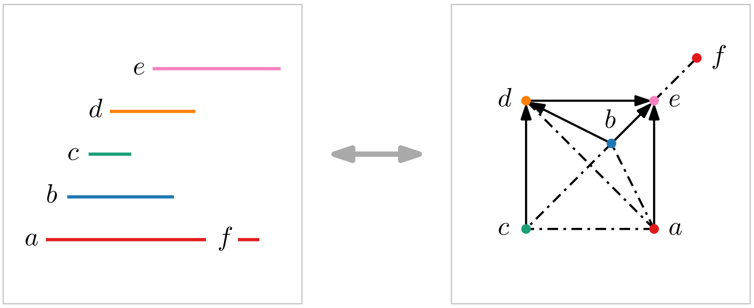


Figure 6.1: Example of an interval representation (left) and the corresponding directional interval graph (right). Intervals are the vertices, and there is an undirected edge (dash dotted) if intervals nest and an arc towards the right interval if the intervals overlap.

some restricted graph classes, e.g., when the underlying graph is a tree, a series-parallel graph, a graph of bounded tree-width, or a bipartite graph [FKŻ08a, FKŻ08b, RdW08]. Mixed graphs and their colorings have also been studied in the context of (quasi-)upward planar drawings [FKP$^+$14, BDP14, BD16], edge orientations [GHS07, BHZ18] and parameterized complexity [Dam19].

Given a proper coloring c of a mixed graph G, there is the dual interpretation to associate a coloring with an orientation of the undirected edges. More precisely, if the task is to orient all edges such that there are no cycles, then c defines a digraph obtained from G by orienting each edge $\{u, v\}$ such that it points towards v if and only if $c(u) < c(v)$. Additionally, if c is a proper coloring with the minimum number of colors, then the corresponding edge orientation minimizes the longest directed path. If G is not a mixed graph, but an undirected graph, this duality is known as the Gallai–Hasse–Roy–Vitaver theorem [Gal68, Has65, Roy67, Vit62].

Interval graphs are a classic subject of algorithmic graph theory whose applications range from scheduling problems to analysis of genomes [Gol80]. Many problems that are NP-hard for general graphs can be solved efficiently for interval graphs. In particular, the chromatic number of (undirected) interval graphs [Gol80] and directed acyclic graphs [HKdW97] can be computed in linear time.

Definitions and Notation. A *mixed interval graph* is a mixed graph G whose underlying graph $U(G)$ is an interval graph. For a set $\mathcal{I}$ of intervals on the real line, the *directional interval graph of* $\mathcal{I}$ is a mixed graph G with vertex set $\mathcal{I}$ where, for every two vertices $u = [l_u, r_u]$, $v = [l_v, r_v]$ with u starting to the left of v, i.e., $l_u \leq l_v$, exactly one of the following conditions holds:

$$\begin{aligned} u \text{ and } v \text{ are disjoint, i.e., } r_u < l_v &\Leftrightarrow u \text{ and } v \text{ are independent in } G, \\ u \text{ and } v \text{ overlap, i.e., } l_u < l_v \leq r_u < r_v &\Leftrightarrow \text{the arc } (u, v) \text{ is in } G, \\ u \text{ contains } v \text{, i.e., } r_v \leq r_u &\Leftrightarrow \text{the edge } \{u, v\} \text{ is in } G. \end{aligned}$$

For an example see Figure 6.1.

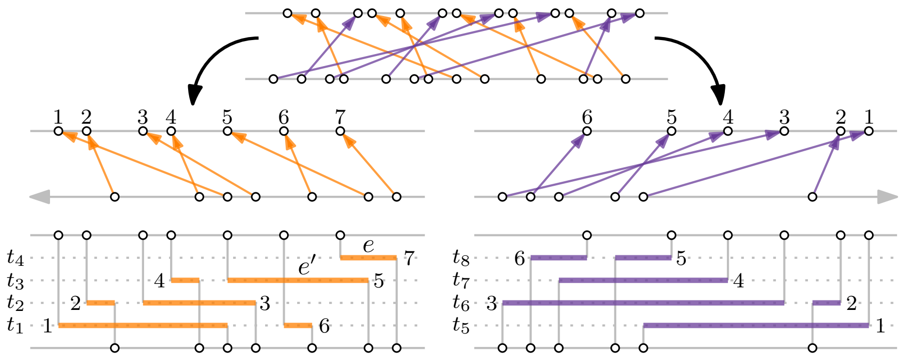


Figure 6.2: Separate greedy assignment of left-going and right-going edges to tracks.

We can also classify a graph as a directional interval graph without having one specific set of intervals as its vertex set. So, a mixed graph G is a *directional interval graph* if G is the directional interval graph of some set of intervals. Consequently, similarly to interval graphs, a directional interval graph may have several different interval representations. Observe that the endpoints in any directional representation can be perturbed so that every endpoint is unique, and the modified intervals represent the same graph. Gutowski et al. [GMR$^+$22] show how to recognize directional interval graphs and how to compute a corresponding interval representation (if one exists) in $\mathcal{O}(n^2)$ time, where n is the number of vertices, by using PQ-trees. Note that, as there is no directed circuit in a directional interval graph G, $\chi(G)$ is always well defined.

Further, we generalize directional interval graphs and directional representations to *bidirectional interval graphs* and *bidirectional representations*. There, we assume that we have two types of intervals, which we call *left-going* and *right-going*. For left-going intervals, the edges and arcs are defined as in directional interval graphs. For right-going intervals, the symmetric definition applies, that is, we have an arc (u, v) if and only if $l_v < l_u \leq r_v < r_u$. Moreover, there is an edge for every pair of a left-going and a right-going interval that intersect.

Motivation. In this chapter, we combine the research directions of coloring geometric intersection graphs and coloring mixed graphs by studying the problem of coloring mixed interval graphs. Aside from being a natural combination of these two research directions, we draw further motivation from the following application in layered graph drawing, which stems from Part II.

A subproblem that occurs within the Sugiyama framework [STT81] is the edge routing step. This step is applied to every pair of consecutive layers. We formalize this step for orthogonal edges as follows. Given a set of points on two horizontal lines (corresponding to vertices or so-called *ports* on two consecutive layers) and a perfect matching between the points on the lower and those on the upper line, connect the matched pairs of points by x- and y-monotone rectilinear paths. Since we assume that no two points have the same x-coordinate, each pair of points can be connected by a path that consists of three axis-aligned line segments – a vertical one, a horizontal one, and another vertical one; see Figure 6.2. We refer to the interval that corresponds to the vertical projection of an edge to the x-axis as the *span* of that edge. For ease of description, we presume that all edges point upward. This allows us to classify each edge either as *left-going* or as *right-going*.

Now the task is to map the horizontal pieces to new horizontal lines in between the two given lines where the edge endpoints are placed. We call these intermediate lines *tracks*; see Figure 6.2. For each pair of horizontal pieces, we require that they do not overlap and do not cross twice. This implies that any two edges whose spans intersect must be mapped to different tracks. If there is a left-going edge e whose span overlaps that of another left-going edge e' that lies further to the left (see e and e' in Figure 6.2), then e must be mapped to a higher track than e' to avoid crossings. The symmetric statement holds for pairs of right-going edges.

The objective is to minimize the number of tracks in order to get a compact layered drawing of the original graph. This corresponds to minimizing the number of colors in a proper coloring of a bidirectional interval graph.

Instead of solving this problem exactly, the following strategy is a natural heuristic. Assign all left-going and all right-going edges to tracks independently and then combine these assignments by drawing the left-going edges below the right-going edges; see the indices of the tracks in Figure 6.2. For this heuristic, it suffices to compute proper colorings of directional interval graphs – once for the left-going edges and once for the mirrored right-going edges. To this end, consider the greedy strategy that assigns each interval (in order by left endpoints) the minimum free color while respecting incoming edges in the corresponding directional interval graph.

Contribution. We first show that this previously described greedy strategy colors directional interval graphs with the minimum number of colors; see Section 6.2. This yields a simple 2-approximation algorithm for the bidirectional case. Then, we prove that computing the chromatic number of a mixed interval graph is NP-hard; see Section 6.3. There, we also show that this result extends to proper interval graphs. We conclude and propose open problems in Section 6.4.

6.2 Coloring Directional Interval Graphs

In this section, we first describe a greedy algorithm formally, which we use in Section 7.3.6 for drawing layered graphs like cable plans (see Chapter 7). Then, we prove that this greedy algorithm computes an optimal coloring for a given directional interval representation of a directional interval graph $G = (V, E, A)$. If we are not given an interval representation (i.e., a set of intervals) but only the directional interval graph G, we obtain a representation of G in quadratic time by the recognition algorithm by Gutowski et al. [GMR$^+$22].

The greedy algorithm generalizes the classical greedy coloring algorithm for (undirected) interval graphs. Also, our optimality proof follows, on a high level, the strategy of relating the coloring to a large clique. In our setting, however, the underlying geometry is more intricate, which makes the optimality proof as well as a fast implementation more involved. The algorithm works as follows; see Figure 6.2 (left) for an example.

Greedy Algorithm. Iterate over the given intervals in increasing order of their left endpoints. For each interval v, assign v the smallest available color $c(v)$. A color k is *available* for v if, for any interval u that has already been colored, $k \neq c(u)$ if u contains v and $k > c(u)$ if u overlaps v.

A naive implementation of the greedy algorithm runs in quadratic time. Using augmented binary search trees, we can speed it up to optimal $\mathcal{O}(n \log n)$ time.

Lemma 6.1. *The greedy algorithm can be implemented to color n intervals in $\mathcal{O}(n \log n)$ time, which is worst-case optimal assuming the comparison-based model.*

Proof. We describe a sweep-line algorithm sweeping from left to right. In a first step, we show how to achieve a running time of $\mathcal{O}(m+n \log n)$, where $n = |V|$ and $m = |A|$. Then we use an additional data structure in order to avoid the $\mathcal{O}(m)$ term in the running time. Note that m can be quadratic in n. For the faster implementation, we do not assume knowledge of G, only knowledge of the set of intervals.

Build a balanced binary search tree $\mathcal{T}$ to keep track of the currently available colors. Initially, $\mathcal{T}$ contains the colors 1 to n. Fill a list $\mathcal{L}$ with the $2n$ endpoints of the intervals in V (which we can assume to be pairwise different). Sort $\mathcal{L}$. Then traverse $\mathcal{L}$ in this order, which corresponds to a left-to-right sweep. There are two types of events for an endpoint $t \in \mathcal{L}$.

Left: If t is the left endpoint of an interval v, let x be the largest color over all intervals that have an arc to v, that is, $x = \max\{c(v) \colon (u,v) \in A\} \cup \{0\}$. Then, search in $\mathcal{T}$ for the smallest color $y > x$, delete y from $\mathcal{T}$, and set $c(v) = y$.

Right: If t is the right endpoint of an interval v, we insert $c(v)$ into $\mathcal{T}$ because $c(v)$ is available again.

Clearly, this implementation runs in $\mathcal{O}(m + n \log n)$ time. To avoid the $\mathcal{O}(m)$ term, we use a second binary search tree $\mathcal{T}'$ that maintains the currently active intervals, sorted according to color. We augment $\mathcal{T}'$ by storing, in every node ν, the leftmost right endpoint r^ν in its subtree. Any interval that contains the current endpoint $t \in \mathcal{L}$ is *active.*

At a Left event, this allows us to determine, in $\mathcal{O}(\log n)$ time, the interval u with the largest color x among all active intervals that overlap the new interval v (that is, $r_u < r_v$), as follows. We find u by descending $\mathcal{T}'$ from its root. From the current node, we go to its right child ρ whenever $r^\rho < r_v$. If such an interval does not exist, we set $x = 0$. Then we continue as above, querying $\mathcal{T}$ for the smallest available color $y > x$. Finally, we set $c(v) = y$ and add v to $\mathcal{T}'$. Observe that we can update the augmented information also in logarithmic time.

At a Right event, we update $\mathcal{T}$ as above. Additionally, we need to update $\mathcal{T}'$. We do this by deleting the interval v that is about to end and at the same time update the augmented information again in logarithmic time.

We now argue that, for outputting the greedy solution of our coloring problem, the running time of $\mathcal{O}(n \log n)$ is worst-case optimal assuming the comparison-based model of computation. Suppose that a coloring algorithm would run in $o(n \log n)$ time. Then, we could use it to sort any set $\{a_1, \dots, a_n\}$ of n numbers in $o(n \log n)$ time by coloring the set $\{[a_1 - M, a_1 + M], \dots, [a_n - M, a_n + M]\}$ of intervals, where $M = \max\{a_1, \dots, a_n\} - \min\{a_1, \dots, a_n\}$. Namely, the corresponding directional interval graph is a tournament graph[13] and for each $i \in \{1, \dots, n\}$, the color of the interval $[a_i - M, a_i + M]$ in an optimal coloring corresponds to the rank of a_i in a sorted version of $\{a_1, \dots, a_n\}$. □

Next, we show that the greedy algorithm computes an optimal proper coloring. This also yields a simple 2-approximation for the bidirectional case.

Theorem 6.2. *Given an interval representation of a directional interval graph G with n intervals, the greedy algorithm computes a proper coloring of G with $\chi(G)$ many colors in $\mathcal{O}(n \log n)$ time.*

Proof. In Lemma 6.1, we have shown that the greedy algorithm described in this section runs in $\mathcal{O}(n \log n)$ time. It remains to argue that the greedy algorithm computes a proper coloring with the minimum number of colors.

The *transitive closure* G^+ of G is the mixed graph that we obtain by exhaustively adding transitive arcs, i.e., if there are arcs (u, v) and (v, w), we add the arc (u, w) if absent. Clearly, no pair of adjacent intervals in the underlying undirected graph $U(G^+)$ of G^+ can have the same color in a proper coloring of G. Therefore, $\omega(U(G^+)) \leq \chi(G)$ where $\omega(U(G^+))$ denotes the size of a largest clique in $U(G^+)$. We show below that the greedy algorithm computes a coloring with at most $\omega(U(G^+))$ many colors, which must therefore be optimal. For $v \in V$ let $\mathcal{I}_{\mathsf{in}}(v)$ be the set of intervals having an arc to v in G.

Let c be the coloring computed by the greedy coloring algorithm. Since we always pick an available color, c is a proper coloring. To prove optimality of c, we show the existence of a clique in $U(G^+)$ of cardinality $c_{\max} = \max_{v \in V} c(v)$.

Consider an interval $v_0 = [l_0, r_0]$ of color $c_{\max}$. Among $\mathcal{I}_{\mathsf{in}}(v_0)$, let v_1 be the unique interval with the largest color (all intervals in $\mathcal{I}_{\mathsf{in}}(v_0)$ have different colors as they share the point l_0). We call v_1 the *step below* v_0. We repeat this argument to find the step v_2 below v_1 and so on. For some $t \geq 0$, there is a v_t without a step below it, namely where $\mathcal{I}_{\mathsf{in}}(v_t) = \emptyset$. We call the sequence $v_0, v_1, \dots, v_t$ a *staircase* and each of its intervals a *step*; see Figure 6.3. Clearly, (v_j, v_i) is an arc of G^+ for $0 \leq i < j \leq t$. In particular, the staircase is a clique of size $t + 1$ in $U(G^+)$. Next, we argue about the intervals with colors between the steps.

For a step $v_i = [l_i, r_i]$, $i \in \{0, \dots, t\}$, let S_i denote the set of intervals that contain the point l_i and have a color $x \in \{c(v_{i+1}) + 1, c(v_{i+1}) + 2, \dots, c(v_i)\}$, where we define $c(v_{t+1}) = 0$; see Figure 6.3. Note that $v_i \in S_i$ and, by the definition of steps, each interval in $S_i \setminus \{v_i\}$ contains v_i. Moreover, observe that $|S_i| = c(v_i) - c(v_{i+1})$, as

[13] A tournament graph is a digraph in which every pair of distinct vertices is connected by one of the two possible arcs. Or in other words, it is an oriented complete graph.

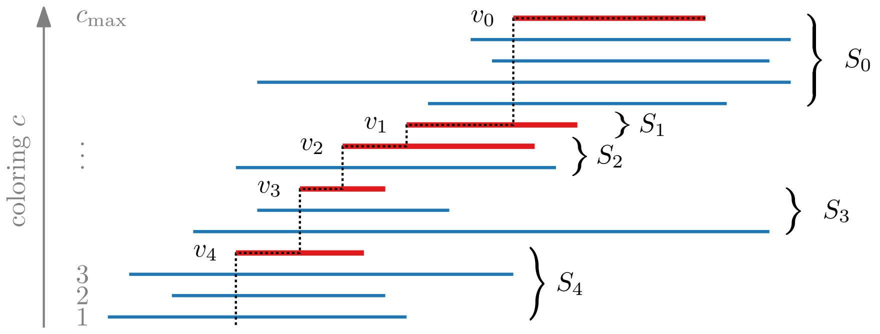
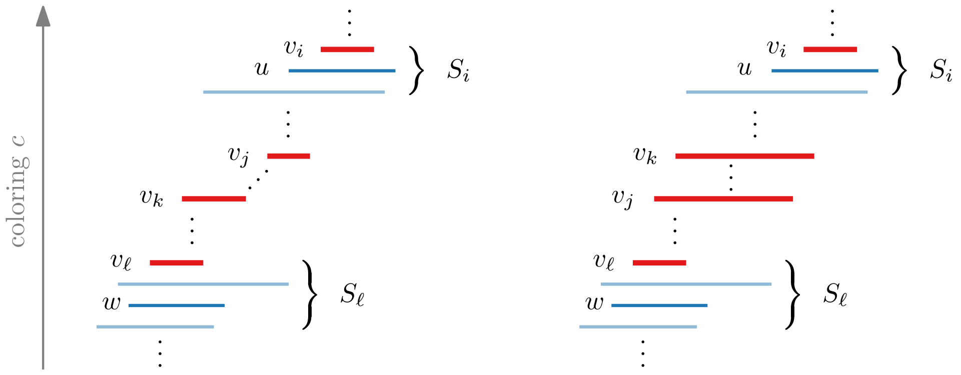


Figure 6.3: A staircase and its intermediate intervals, which form a clique in $U(G^+)$.



(a) arcs (w, v_k), (v_k, v_j), and $(v_j, u,)$ exist in G^+ **(b)** arcs (w, v_j) and $(v_j, u,)$ exist in G^+

Figure 6.4: Given any two intervals $u \in S_i$ and $w \in S_\ell$ with $i < \ell$, there is a directed path from w to u in G or an edge $\{w, u\}$ in G. Therefore, the edge uw is present in $U(G^+)$.

otherwise, the greedy algorithm would have assigned a smaller color to v_i. It follows that $c_{\max} = \sum_{i=0}^{t} |S_i|$.

We claim that $S = \bigcup_{i=0}^{t} S_i$ is a clique in $U(G^+)$. Let $u \in S_i$, $w \in S_\ell$ such that $u \cap w = \emptyset$ (otherwise they are clearly adjacent in $U(G^+)$). Assume without loss of generality that $i < \ell$; see Figure 6.4. Let j, k be the largest and smallest index so that $v_j \cap u \neq \emptyset$ and $v_k \cap w \neq \emptyset$, respectively. Observe that $u \cap w = \emptyset$, $u \cap v_{i+1} \neq \emptyset$, and $w \cap v_{\ell-1} \neq \emptyset$ imply $i < j < \ell$ and $i < k < \ell$. Since u does not intersect v_{j+1}, it overlaps with v_j, i.e., G contains the arc (v_j, u) and likewise, since w does not intersect v_{k-1}, it overlaps with v_k, i.e., G contains the arc (w, v_k).

If $j < k$ (see Figure 6.4a), then G^+ contains (w, v_k) and (v_k, v_j), and therefore (w, v_j). If $j \geq k$ (see Figure 6.4b), then v_j is adjacent to both u and w, and since u, w are disjoint, v_j overlaps with u and w, i.e., G contains (w, v_j). In either case, the presence of (w, v_j) and (v_j, u) implies that G^+ contains (w, u). It follows that S forms a clique in $U(G^+)$. $\square$

We remark that our proof that the chromatic number equals the clique number for $U(G^+)$, which is the underlying undirected graph of the transitive closure of G, is a necessary condition for $U(G^+)$ to be perfect. To be a perfect graph, this property must also hold for every induced subgraph. It has been shown by Brückner [Brü21] that $U(G^+)$ is indeed perfect.

As mentioned before, Theorem 6.2 directly yields a simple 2-approximation for the bidirectional case. The main idea is to solve the problem for left-going and right-going intervals independently and then to stack the one drawing on top of the other.

Corollary 6.3. *Given a bidirectional interval representation of n intervals, there is an $\mathcal{O}(n \log n)$-time algorithm that computes a 2-approximation of an optimal proper coloring of the corresponding bidirectional interval graph.*

Proof. Let $\mathcal{I}$ be the set of intervals of G. We split $\mathcal{I}$ into a set of left-going intervals $\mathcal{I}_1$ and into a set of right-going intervals $\mathcal{I}_2$. These sets induce the directional interval graphs G_1 and G_2, respectively.[14] Now we color G_1 and G_2 independently with our greedy coloring algorithm according to Theorem 6.2, and we re-combine them by using different sets of colors for G_1 and G_2. All steps can be performed in $\mathcal{O}(n \log n)$ time. This is a proper coloring of G with $\chi = \chi(G_1) + \chi(G_2)$ colors since between any interval in $\mathcal{I}_1$ and any interval in $\mathcal{I}_2$, there may be an edge but no arc. Clearly, $\chi \leq 2 \max\{\chi(G_1), \chi(G_2)\} \leq 2\chi(G)$, which concludes the proof. □

6.3 Coloring Mixed Interval Graphs

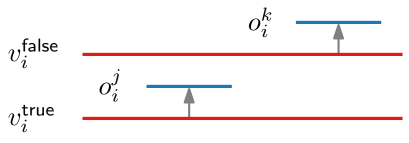
In this section, we show that computing the chromatic number of a mixed interval graph is NP-hard. Recall that the chromatic number can be computed efficiently for interval graphs [Gol80], directed acyclic graphs [HKdW97], and directional interval graphs (Theorem 6.2). In other words, coloring interval graphs becomes NP-hard only if edges and arcs are combined in a non-directional way.

Theorem 6.4. *Given a mixed interval graph G and a number k, it is NP-complete to decide whether G admits a proper coloring with at most k colors.*

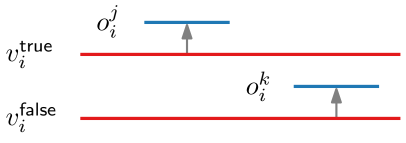
Proof. Containment in NP is clear since a specific proper coloring with k colors serves as a certificate of polynomial size. We prove NP-hardness by a polynomial-time reduction from 3-SAT, which is the problem to decide whether a Boolean formula in conjunctive normal form with at most three literals per clause is satisfiable – w.l.o.g. we assume that every clause has exactly three distinct literals.

The high-level idea is as follows. We are given a 3-SAT formula Φ with variables $v_1, v_2, \ldots, v_n$, and clauses $c_1, c_2, \ldots, c_m$. A literal is a variable or a negated variable – we refer to them as a *positive* or a *negative* occurrence of that variable. From Φ, we construct in polynomial time a mixed interval graph G_Φ with the property that Φ is satisfiable if and only if G_Φ admits a proper coloring with $6n$ colors.

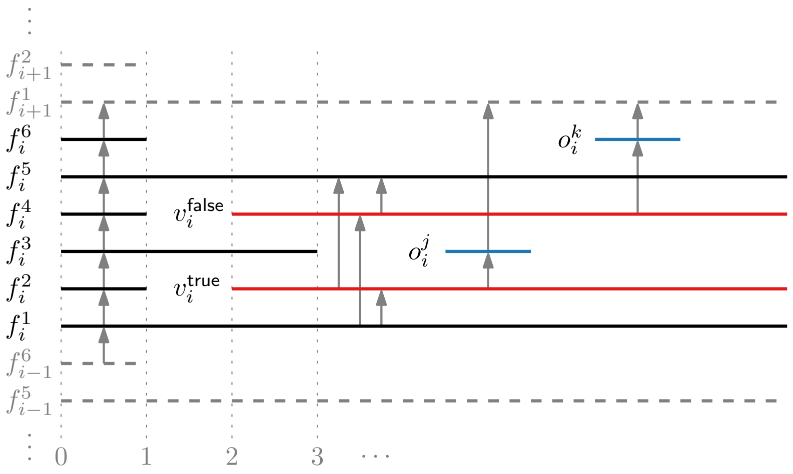
[14] We can mirror the intervals in $\mathcal{I}_2$ along a vertical axis to obtain the directional interval graph G_2 in the usual definition.



(a) variable v_i is true



(b) variable v_i is false



(c) frame (black; intervals starting at position 0) together with a variable gadget (red and blue)

Figure 6.5: A variable gadget for a variable v_i, where v_i appears positively in a clause c_j and negatively in a clause c_k.

To prove that G_Φ is a mixed interval graph, we present an interval representation of $U(G_\Phi)$ and specify which pairs of intersecting intervals are connected by a directed arc, assuming that all other pairs of intersecting intervals are connected by an edge. The graph G_Φ has the property that the color of many of the intervals is fixed in every proper coloring with $6n$ colors. In our figures, the x-dimension corresponds to the real line that contains the intervals, whereas we indicate its color by its position in the y-dimension – thus, we also refer to a color as a *layer*. In this model, our reduction has the property that Φ is satisfiable if and only if the intervals of G_Φ admit a drawing that fits into $6n$ layers.

Our construction consists of a *frame* together with n *variable gadgets* and m *clause gadgets*. Each variable gadget is contained in a horizontal strip of height 6 that spans the whole construction, and each clause gadget is contained in a vertical strip of width 4 and height $6n$. The strips of the variable gadgets are pairwise disjoint, and likewise, the strips of the clause gadgets are pairwise disjoint.

Frame. See Figure 6.5c. The frame consists of six intervals $f_i^1, f_i^2, \dots, f_i^6$ for each of the variables v_i, where $i \in \{1, \dots, n\}$. All of these intervals start at position 0 and extend from the left into the construction. The intervals f_i^2, f_i^4, f_i^6 end at position 1. The intervals f_i^1 and f_i^5 extend to the very right of the construction. Interval f_i^3 ends at position 3. Further, there are arcs (f_i^j, f_i^{j+1}) for each $j \in \{1, \dots, 5\}$ and (f_i^6, f_{i+1}^1) for each $i \in \{1, \dots, n-1\}$. This structure guarantees that any proper coloring with colors $\{1, 2, \dots, 6n\}$ assigns color $6(i-1)+j$ to interval f_i^j.

Variable Gadget. See Figure 6.5. For each variable v_i, where $i \in \{1, \dots, n\}$, we have two intervals v_i^{false} and v_i^{true}, which start at position 2 and extend to the very right of the construction. Moreover, they both have an incoming arc from f_i^1 and an outgoing arc to f_i^5. This guarantees that they are drawn in the layers of f_i^2 and f_i^4, however, their ordering can be chosen freely. We say that v_i is set to true if v_i^{true} is below v_i^{false}, and v_i is set to false otherwise.

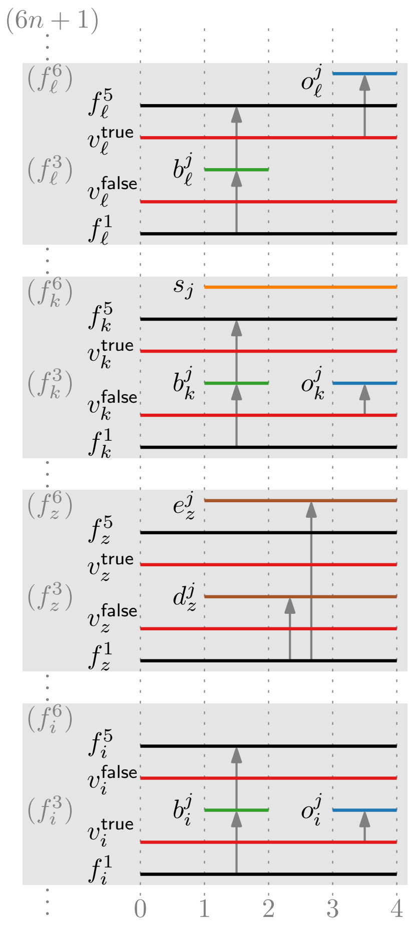
For each occurrence of v_i in a clause c_j, where $j \in \{1, \dots, m\}$, we create an interval o_i^j within the clause gadget of c_j. There is an arc $(v_i^{\mathsf{true}}, o_i^j)$ for a positive occurrence and an arc $(v_i^{\mathsf{false}}, o_i^j)$ for a negative occurrence as well as an arc (o_i^j, f_{i+1}^1) if $i < n$. This structure guarantees that o_i^j is drawn either in the same layer as f_i^3 or as f_i^6. However, drawing o_i^j in the layer of f_i^3 (which lies between v_i^{true} and v_i^{false}) is possible if and only if the chosen truth assignment of v_i satisfies c_j.

Clause Gadget. See Figure 6.6. Our clause gadget starts at position $4j$, relative to which we describe the following positions. Consider a fixed clause c_j that contains variables v_i, v_k, v_ℓ. We create an interval s_j of length 3 starting at position 1. The key idea is that s_j can be drawn in the layer of f_i^6, f_k^6 or f_ℓ^6, but only if o_i^j, o_k^j or o_ℓ^j, each of which has length 1 and starts at position 3, is not drawn there. This is possible if and only if the corresponding variable satisfies the clause.

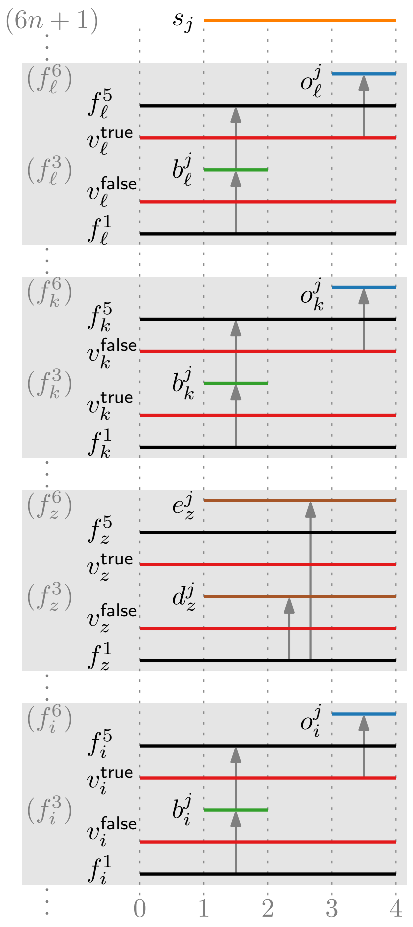
To ensure that s_j does not occupy any other layer, we block all the other layers. More precisely, for each variable v_z with $z \notin \{i, k, \ell\}$, we create *dummy intervals* d_z^j, e_z^j of length 3 starting at position 1 that have an arc from f_z^1 and an arc to f_{z+1}^1. These arcs force d_z^j, e_z^j to be drawn in the layers of f_z^3 and f_z^6, thereby ensuring that s_j is not placed in any layer associated with the variable z.

Similarly, for each $z \in \{i, k, \ell\}$, we create a *blocker* b_z^j of length 1 starting at position 1 that has an arc from f_z^1 and an arc to f_z^5. This fixes b_z^j to the layer of f_z^3 (since the layers of f_z^2 and f_z^4 are occupied by v_z^{true} and v_z^{false}), thereby ensuring that, among all layers associated with v_z, s_j can only be drawn in the layer of f_z^6.

Correctness. Consider for each clause c_j with variables v_i, v_k, and v_ℓ the corresponding clause gadget. To achieve a total height of at most $6n$, s_j needs to be drawn in the same layer as some interval of the frame. Due to the presence of the dummy intervals, the only available layers are the ones of f_z^6 for each $z \in \{i, k, \ell\}$. However, the layer of f_z^6 is only free if o_z^j is not there, which is the case if and only if o_z^j is drawn in the layer of f_z^3. By construction, this is possible if and only if the variable v_z is in the state that satisfies clause j. Otherwise, we need an extra $(6n+1)$-th layer. Both situations are illustrated in Figure 6.6. Hence, $6n$ layers are sufficient if and only if the variable gadgets represent a truth assignment that satisfies all the clauses of Φ. The mixed interval graph G_Φ has polynomial size and can be constructed in polynomial time. □



(a) c_j is satisfied.



(b) c_j is not satisfied.

Figure 6.6: A clause gadget for a clause $c_j = v_i \vee \neg v_k \vee v_\ell$, where $z \notin \{i, k, \ell\}$.

A *proper interval graph* is an interval graph that admits an interval representation of the underlying graph in which none of the intervals properly contains another interval. The class of proper interval graphs equals the class of *unit interval graphs*, which admit an interval representation where every interval has unit length. They are also known under the term *indifference graph.* Clearly, by stretching intervals locally and moving intervals slightly, one can transform a proper interval representation to a unit interval representation of the same graph and vice versa. Proper interval graphs have applications in mathematical psychology [Rob70], where they arise from utility functions, and in DNA sequence assembly in bioinformatics [GGKS95].

We can slightly adjust the reduction presented in the proof of Theorem 6.4 to make G_Φ a *mixed* proper interval graph.

Corollary 6.5. *Given a mixed proper interval graph G and a number k, it is* NP*-complete to decide whether G admits a proper coloring with at most k colors.*

Proof. The general idea is as follows. We start the construction with the same set of intervals as in the proof of Theorem 6.4. Then, we set $x_{\mathsf{left}} = 0$ and $x_{\mathsf{right}} = 4(m+1)$, which is on the right side of all intervals. Next, we describe a procedure that extends every interval so that it has the left endpoint in x_{left} or has the right endpoint in x_{right}. The procedure adds some new intervals and merges some groups of intervals into one interval. The total height of the interval representation increases to $4n + 2nm$ during the procedure. Finally, we extend every interval at x_{left} (x_{right}) to the left (right) by a length inverse to its current total length (wherever we have ties, we slightly perturb the intervals). These extensions guarantee that in the end, no interval contains another interval. In the remainder of this proof, we describe the procedure of extending, adding, and merging intervals. For an illustration, see Figure 6.7.

The intervals of the frame and all v_i^{true} and v_i^{false} with $i \in \{1, \dots, n\}$ already end at x_{left} or x_{right}. Currently, in any drawing of G_Φ with $6n$ layers and for every fixed $i \in \{1, \dots, n\}$, all the intervals b_i^j, and o_i^j with $j \in \{1, \dots, m\}$ are drawn in the layers of f_i^3 and f_i^6. Additionally, each dummy interval and each s_j is drawn in one of these layers. We divide these layers into m copies each so that each pair of b_i^j and o_i^j has its own two layers. This copy process works as follows.

First, we replace each f_i^3 and f_i^6 by m copies each. Accordingly, we adjust the height of the drawing to be $4n + 2nm$. Then, we make m copies of each dummy interval and virtually assign each copy to a distinct layer of the drawing. For each b_i^j, we virtually assign it to the layer of the j-th copy of f_i^3 and extend it to the left up to x_{left}. In this process, we merge b_i^j with every dummy interval on its left and with the j-th copy of f_i^3 while keeping all involved arcs. We call the merged interval $f_i^{3,j}$. If there is no b_i^j in some of these layers, we obtain $f_i^{3,j}$ by extending the j-th copy of f_i^3 up to x_{right} and merging it with all dummy intervals virtually assigned to its layer.

Symmetrically to b_i^j, we extend each o_i^j to the right up to x_{right} and merge o_i^j with all dummy intervals virtually assigned to the layer of $f_i^{3,j}$, but here we drop the arcs of the dummy intervals. We call the merged interval o'^j_i. Similarly, for every clause c_j with variables v_i, v_k, v_ℓ, we merge all dummy intervals virtually assigned to the layer of the j-th copy of f_z^6, for each $z \in \{i, k, \ell\}$ that are to the right of s_j

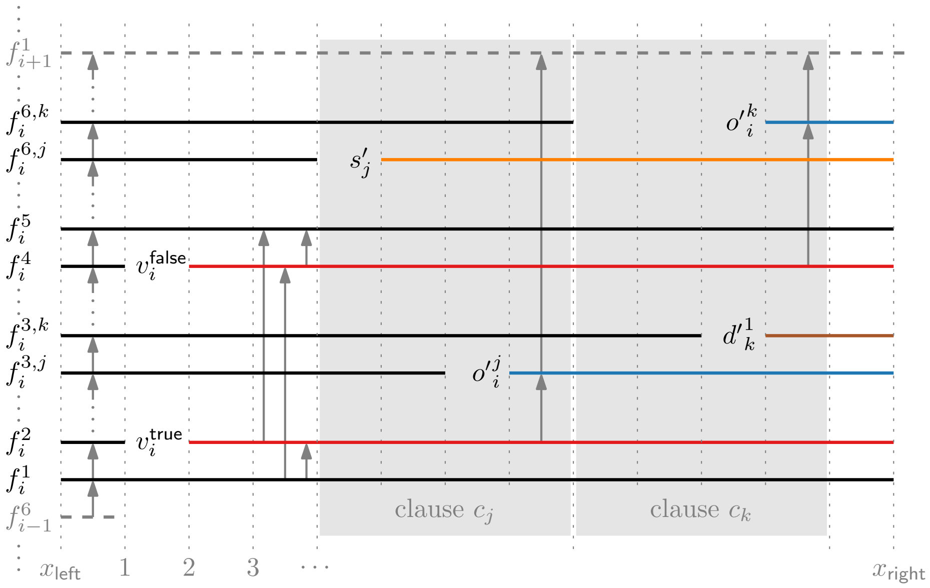


Figure 6.7: Our modifications on our NP-hardness construction to use only proper intervals. For a variable gadget of a variable v_i, we replace the layers of f_i^3 and f_i^6 by m new layers each – one for each clause. Here, the new layers of the clauses c_j and c_k, which contain v_i, are depicted.

and drop all the arcs as in the previous case. We obtain three copies d'^1_j, d'^2_j, and d'^3_j of the same interval and we merge one of these copies, say d'^3_j, with s_j. We denote that new interval by s'_j. We have dropped all arcs of d'^1_j, d'^2_j, and s'_j to preserve the freedom we had for placing s_j in our original construction. The only unmerged dummy intervals are in the layer of the j-th copy of f_i^6 to the left of s_j or in the layer of the j-th copy of f_i^6 if there is no occurrence of the variable v_i in the clause c_j. In each of these layers, we merge the dummy intervals together with the corresponding copy of f_i^6 while keeping all involved arcs and obtain intervals ending at x_{left}. For each $j \in \{1, \dots, m\}$, we call the merged interval $f_i^{6,j}$.

For each pair of $i \in \{1, \dots, n\}$ and $j \in \{1, \dots, m-1\}$, we add the arcs $(f_i^2, f_i^{3,1})$, $(f_i^{3,j}, f_i^{3,j+1})$, $(f_i^{3,m}, f_i^4)$, $(f_i^5, f_i^{6,1})$, $(f_i^{6,j}, f_i^{6,j+1})$, and $(f_i^{6,m}, f_{i+1}^1)$ to have a frame as in the original hardness construction. Observe that this new frame has exactly $4n + 2nm$ intervals with their left endpoint at x_{left} and, in the whole construction, there are $2n + 6m$ other intervals with their right endpoint at x_{right}, i.e., the $2n$ intervals v_i^{true} and v_i^{false}, where $i \in \{1, \dots, n\}$ and the $6m$ intervals o'^j_i, d'^1_j, d'^2_j, and s'_j, where $j \in \{1, \dots, m\}$ and i corresponds to the three variables contained in c_j.

Next, we argue that the functionality described in the proof of Theorem 6.4 is retained. Intervals of the (new) frame either block a complete layer from x_{left} to x_{right}, or they end at position 1 (each f_i^2 and f_i^4) or within the clause gadget of a clause c_j if the considered variable v_i occurs in c_j (each $f_i^{3,j}$ and $f_i^{6,j}$). Any other interval starting in a clause gadget of a clause c_j needs to be matched with a frame interval that ends in the clause gadget of c_j. Therefore, to have a construction with a total height of at most $4n + 2nm$, we need to combine $f_i^{3,j}$ and $f_i^{6,j}$ with o'^{j}_{i} and some of $\{d'^{1}_{j}, d'^{2}_{j}, s'_j\}$, while $f_i^{3,j}$ and s'_j are not compatible. This ensures that the correctness argument from the proof of Theorem 6.4 remains valid. □

6.4 Concluding Remarks and Open Problem

We have introduced the natural concept of directional interval graphs as a generalization of classical interval graphs by taking containment, overlap, and left-to-right-ordering in an interval representation into account. In a further generalization, we assume that in a set of intervals, we have two types of intervals. Namely, each interval has a given internal binary state (which we have called *left-going* or *right-going*). This state additionally affects the edge directions. We denote the resulting graphs as bidirectional interval graphs.

Coloring directional interval graphs is of special interest as it corresponds to a direction-consistent layering of a corresponding interval representation. For bidirectional interval graphs, this is equivalent to the problem of finding a minimum number of tracks for orthogonal layered graph drawing (see Chapter 7) under the restriction that double crossings are forbidden and if we exclude special cases that can occur if we have non-distinct x-coordinates for the endpoints of the edges. In Section 7.3.6, we describe how we also handle these special cases in practice.

Directional interval graphs preserve sufficiently much structure such that finding a minimum proper coloring remains polynomial-time solvable as it is for purely undirected and purely directed interval graphs. For bidirectional interval graphs, we at least obtain a 2-approximation.

In contrast to directional interval graphs, we have seen that the problem of finding a minimum-size proper coloring becomes NP-complete for (general) mixed interval graphs, i.e., interval graphs where we can freely assign undirected edges or directed edges with arbitrary edge direction. For this hardness to occur, we need to have both undirected edges and arcs simultaneously. Our NP-hardness reduction uses a linear number of arcs and a quadratic number of undirected edges. Where does it become hard? Clearly, by guessing edge directions, the problem is in XP in the number of undirected edges. Is it also in FPT? What about a small number of directed edges?

The recognition problem, which we have not investigated here, also leaves some gaps. Gutowski et al. [GMR+22] present an algorithm with runtime quadratic in the number of vertices. Can we recognize directional interval graphs also in linear time?

Moreover, bidirectional graphs are not yet fully understood for the questions considered in this chapter. Can we recognize bidirectional interval graphs in polynomial time? Can we color bidirectional interval graphs optimally in polynomial time, or at least find α-approximate solutions with $\alpha < 2$?

There is another natural definition for oriented interval graphs, where we have a directed edge for containing intervals and an undirected edge for overlapping intervals (inverse to our definition). It would also be interesting to investigate this setting.

Part II

Drawing Graphs – Applied Results

Chapter 7
Layered Drawing of Undirected Graphs with Port Constraints

In this book, the part for applied graph drawing consists of only this chapter, which, however, tackles a basic problem in our modern industrialized world: *how to draw a technical plan automatically.*

There is a large variety of technical plans and drawing styles for technical plans, which makes it almost impossible to design a single algorithm to draw all these plans automatically. Of course, using artificial intelligence and a large training data set may in the long run solve this problem. Then, however, we cannot understand the internal processes and adjust them easily. We also need to rely on the trained model where we may not be able to guarantee specific properties of the drawing. Moreover, gaining a large suitable training data set is another difficult task. Hence, our goal is to design a clear and understandable algorithm based on theoretic foundations that is applicable in specific technical domains.

7.1 Introduction

Today, the development of industrial machinery implies a high interdependency of mechanical, electrical, hydraulic, and software-based components. The continuous improvement of these machines yields an increased complexity within their components, but also in their interrelations.

In the case of a malfunction, a human technician needs to understand the particular interdependencies. Only then, it is possible to find, understand, and resolve errors. Different types of schematics play a key role in this diagnosis task for depicting dependencies between the involved components, e.g., electric or functional schematics. The intuitive understanding and the comprehensibility of these schematics are critical for finding errors effectively and efficiently.

Due to the increased complexity of machinery, such schematics cannot be drawn manually anymore. Also, the high variance of machine configurations and their combinations leads to large number of (slightly) different plans. Suppose a car manufacturer has three series of cars with four types of drives (gasoline, diesel, electric, hybrid), each with six levels of power, and four optional extra equipment packages. Already in this oversimplified model, this results in 1152 different combinations that need to be drawn.

Creating manual drawings on demand is also not practically doable since technicians often do not know in forehand which plans they need and customers want their

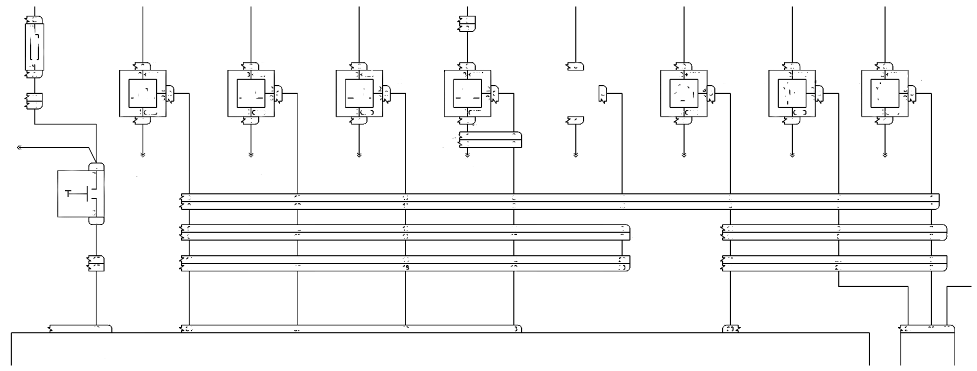


Figure 7.1: Extract of a hand-drawn plan. The labels have been intentionally obfuscated.

machines to be repaired fast and neatly. Therefore, an automatic ad-hoc visualization of schematics appropriate for the requested diagnosis case is required.

To support technicians, algorithms for drawing schematics should adhere to the visual "laws" of the manual drawings that the technicians are familiar with; see Figure 7.1 for an example. Such drawings route connections between components in an orthogonal manner. Manual drawings often use few layers and seem to avoid crossings and bends as much as possible.

Previous Work. When visualizing data flow diagrams or hierarchies, connections are usually directed from left to right or from top to bottom. This setting is supported by the framework introduced by Sugiyama, Tagawa, and Toda in 1981 [STT81]. Given a directed graph, the edges are arranged mainly in the same direction by organizing the nodes in subsequent layers (or levels). The layer-based approach solves the graph-layout problem by dividing it into five phases: cycle elimination, layer assignment, crossing minimization, node placement, and edge routing.

In many technical drawings (such as cable plans or UML diagrams), components are drawn as axes-aligned rectangles, connections between the components are drawn as axes-aligned polygonal chains that are attached to a component using a *port*, that is, the specific point where an edge enters the rectangle of a component. Often, there are extra port labels placed close to a port. Also, the port itself may be drawn as a geometric icon (e.g. a solid square) that is small relative to its component and attached to the boundary of that component. A port has a specific meaning for the domain expert, e.g., it is a pin for a wire at a device. Using so-called *port constraints*, a user can insist that a connection enters a component on a specific side – a natural requirement in many applications. There are further port constraints like *port groups* and *port pairings*. Within a port group, ports can be drawn in arbitrary order to improve the aesthetics of the layout, but together the ports of a group must form a contiguous block because they model, e.g., plug sockets. In a port pairing, two ports need to be drawn on opposite sides of their component and on an axis-aligned line because they model, e.g., a plug that connects pairs of wires.

In 2014, Schulze, Spönemann, and von Hanxleden [SSvH14] have investigated the Sugiyama framework in combination with ports and port constraints. Their resulting implementation is the now well-established *Kieler* library [kie, elk20]. Kieler is particularly interesting for our application as Kieler allows the user to specify some types of port constraints; namely, on which side of a vertex rectangle should a port be placed, and, for each side, the exact order in which the ports should be arranged. Alternatively, the order is variable and can be exploited to improve the layouts in terms of crossings and bends. Okka, Dogrusoz, and Balci [ODB21] integrate these types of port constraints to a force-directed layouting algorithm.

There are also algorithms for practical applications purely based on the orthogonal drawing paradigm, where all vertices are rectangles on a regular grid and the edges are routed along the horizontal and vertical lines of the grid. Here, a classic three-phase method dates back to Biedl, Madden, and Tollis [BMT00]. Chimani et al. [CGMW10, CGM$^+$10] keep the upward-planar-drawing paradigm but avoid strict vertex layering. They can also handle hypergraphs and ports [CGM$^+$10].

We have chosen to build our algorithm for undirected graphs on the (directed) layer-based approach instead of an (undirected) purely orthogonal one because the typical hand-drawn plans use only few distinct layers to place the vertices on, the layer-based approach seems to be better investigated in practice, and Kieler has already proven to yield by and large pleasing results in the considered domain.

Contribution. First, we propose two methods to direct the edges of the given undirected graph so that we can apply the Sugiyama framework (see Section 7.3); one is based on breadth-first search, the other on a force-directed layout. We compare the two methods experimentally with a simple baseline method that places the nodes of the given graph randomly and directs all edges upward (see Section 7.4.4), both on real-world and synthetic cable plans (see Section 7.4.2). We claim that our approach to generate realistic test graphs is of independent interest. We "perturb" real-world instances such that, statistically, they have similar features as the original instances.

Second, we extend the set of port constraints that the aforementioned Kieler library allows the user to specify. In order to model plug sockets, we introduce port groups. Within a group, the position of the ports is either fixed or variable. In either case, the ports of a group must form a contiguous block. Port groups can be nested. If the order of a port group is variable, our algorithm exploits this to improve the aesthetics of the layout.

Apart from such hierarchical constraints, we also give the user the possibility to specify port pairings between ports that belong to opposite sides of a vertex rectangle. Such a pairing constraint enforces that the two corresponding ports are placed at the same x- or z-coordinates on opposite sides of the vertex rectangle. Pairing constraints model plugs that are pairs of sockets of equal width plugged into each other.

After formally defining the problem (Section 7.2), we describe our algorithm (Section 7.3). Then, we present our experimental evaluation (Section 7.4) and the discussion thereof (Section 7.5). Finally, we close this chapter with a collection of cable plan drawings computed by our algorithm (Section 7.7).

7.2 Problem Definition

We define the problem Layered Graph Drawing with Generalized Port Constraints as follows. For an illustration refer to Figure 7.3b.

Given: An undirected *port graph* G, which is a 5-tuple (V, P, PG, PP, E), where

- V is the set of vertices – each vertex v is associated with two positive numbers $w(v)$ and $h(v)$; v will be represented by a rectangle of width at least $w(v)$ and height at least $h(v)$ (to ensure a given vertex label can be accommodated),
- P is the set of ports – each port belongs either directly to a vertex or indirectly through a port group (or a nested sequence of port groups),
- PG is the set of port groups – each port group belongs to a side (Top, Bottom, Left, Right, Free) of exactly one vertex and contains a set of ports and port groups (not contained in another port group) whose order is fixed or variable,
- PP is the set of port pairings – each port pairing consists of two unique ports from P that belong to the same vertex (directly or via port groups), and
- E is the set of edges – each edge connects two unique ports from P that are contained in different vertices (there is at most one edge per port), and
- the graph where all ports are contracted into their vertices is connected.

Find: A drawing of G such that

- no drawing elements overlap each other except that edges may cross each other in single points,
- each vertex $v \in V$ is drawn as an axis-aligned rectangle of width at least $w(v)$ and height at least $h(v)$ on a horizontal layer,
- each port $p \in P$ is drawn as a (small, fixed-size) rectangle attached to the boundary of its vertex rectangle (on the specified side unless set to Free),
- when walking along the boundary of a vertex, the ports of a port group (or subgroup) form a contiguous block; and for a port group with fixed order, its ports and port groups appear in that order,
- for each port pair $\{p, p'\} \in PP$, ports p and p' are drawn on the same vertical or horizontal line on opposite sides of their vertex,
- each edge $\{p, p'\} \in E$ is drawn as a polygonal chain of axis-aligned line segments (*orthogonal polyline*) that connects the drawings of p and p', and
- the total number of layers, the width of the drawing, the lengths of the edges, and the number of bends are kept reasonably small.

We have chosen this problem definition to be both simple and extendable to more complex settings by using the described elements as building blocks. For instance, if there are multiple edges per port, then in a preprocessing we can assign each edge its own port and keep them together using a port group. In a post-processing, we draw just one of these ports and we re-draw the ends of the edges incident to the other ports of this group. Or if there are bundles of edges (e.g. a cable with twisted wires), we can keep their ports together by introducing port groups.

Note that our problem definition generalizes the LAYERED GRAPH DRAWING problem that is formalized and solved heuristically by the Sugiyama framework [STT81]. Several subtasks of the framework correspond to NP-hard optimization problems such as ONE-SIDED CROSSING MINIMIZATION [EW94], which is the problem of minimizing the number of crossings in a drawing of a bipartite graph by permuting the vertices of only one partition, where all vertices of the same partition lie on a common line and the resulting two lines are parallel. Hence, we have to make do with a heuristic for our problem, too. We present this heuristic next.

7.3 Algorithm

We assume that we are given a graph as described in Section 7.2. (Otherwise, we can preprocess accordingly.) Similarly to the algorithm by Sugiyama et al. [STT81], our algorithm proceeds in the following phases, which we treat in the next subsections. For a small but complete example, see Figure 7.2.

Phase 1: *Orienting undirected edges.* We orient the undirected edges by drawing the underlying simple graph with a force-directed graph drawing algorithm and then direct all edges upwards. Alternatively, we may orient the edges by a breadth-first search in order of discovery. (Section 7.3.1)

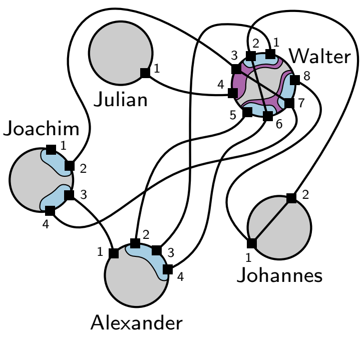
Phase 2: *Assigning vertices to layers.* (Section 7.3.2)

Phase 3: *Orienting ports and inserting dummy vertices.* We try to place a port such that it is on the upper side of its vertex if its incident edge goes upwards and is on the lower side otherwise. However, due to port groups, port pairings, and input constraints, a port may end up on the "wrong" side of its vertex. In this case, we subdivide the incident edge by a dummy vertex on a neighboring intermediate layer to turn the edge direction. (Section 7.3.3)

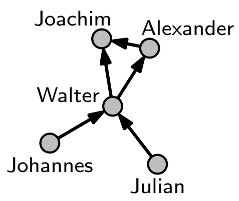
Phase 4: *Reducing crossings by swapping vertices and ports.* We employ the classic barycenter heuristic by Sugiyama et al. [STT81] on a port-wide level to reduce the number of edge crossing. (Section 7.3.4)

Phase 5: *Determining vertex coordinates.* We transform our vertices to ports and apply the algorithm by Brandes and Köpf [BK02, BWZ20] purely on the resulting port structure. (Section 7.3.5)

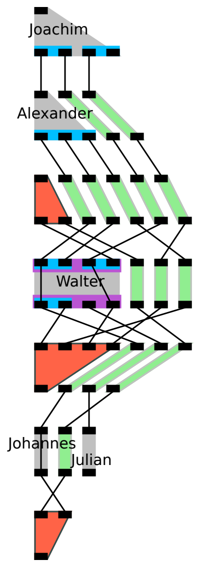
Phase 6: *Constructing the drawing.* We resolve dummy ports and dummy vertices, and we route the edges orthogonally. (Section 7.3.6)



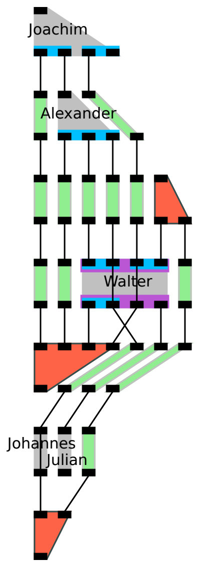
(a) The input graph



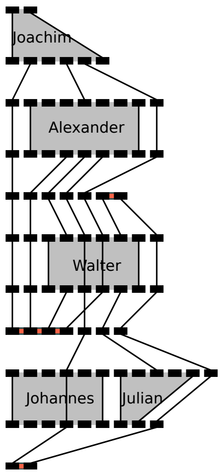
(b) Phase 1: Orienting undirected edges using a force-directed graph drawing algorithm.



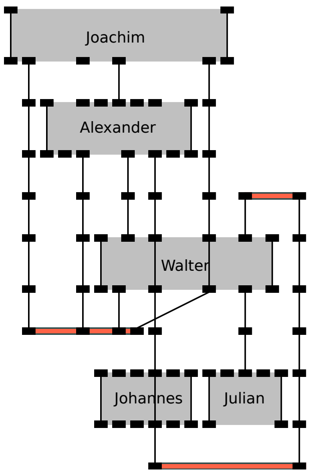
(c) Phases 2 and 3: Assigning vertices to layers, orienting ports, and inserting dummy vertices.



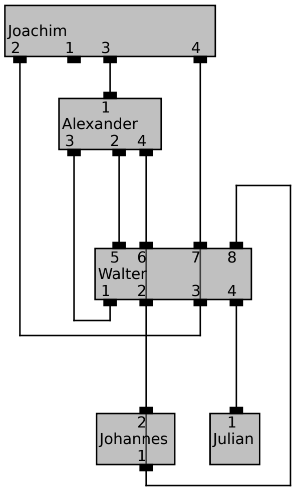
(d) Phase 4: Reducing crossings by swapping vertices and ports.



(e) Phase 5.1: Transforming the drawing to a pure port structure.



(f) Phase 5.2: Determining vertex coordinates by aligning adjacent ports vertically.



(g) Phase 6: Constructing the drawing and routing the edges orthogonally. This is our final drawing.

Figure 7.2: A full example for our algorithm. Port groups are depicted in light blue and violet. (Vertex Walter has nested port groups.) Port pairings are indicated by straight-line segments inside vertices. Dummy vertices are green (for long edges) and red with a frame (turning dummy vertices).

7.3.1 Orienting Undirected Edges

Classical algorithms for layered graph drawing expect as input a directed acyclic graph, whose vertices are placed onto layers such that all edges point downwards. For directed cyclic graphs, some edges may be reversed or removed to make the graph acyclic. In our case of undirected graphs, we suggest the following procedures to orient the undirected edges, making the graph simultaneously directed and acyclic. (Hence, we do not need the cycle elimination phase of the Sugiyama framework.) We ignore the ports in this step.

BFS: We execute a breadth-first search from a random start vertex. Edges are oriented from vertices discovered earlier to vertices discovered later.

FD: We run a force-directed graph drawing algorithm. In the resulting drawing, edges are oriented upwards.

RAND: We place the vertices randomly into the drawing area, uniformly distributed. In the resulting drawing, we orient the edges as in FD.

The runtime of this phase is dominated by the force-directed algorithm. One might consider executing the force-directed algorithm more than once, say k times, with different random start positions and then use the drawing admitting the fewest crossings. This is less time consuming than re-iterating the whole algorithm. Note, however, that it is not clear whether a drawing with fewer crossings is a much better starting point for the rest of the algorithm and justifies the longer running time when choosing $k > 1$. This question may be investigated in new experiments – we have always set $k = 1$.

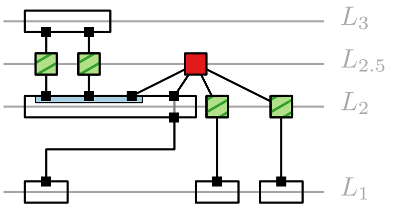
In our experiments, we used a classical spring embedder [FR91] with the speed-up technique as described by Lipp et al. [LWZ16]. The resulting runtime is in $\mathcal{O}(k \cdot I \cdot |V| \log |V|)$, where I is the number of iterations per execution of the force-directed algorithm.

7.3.2 Assigning Vertices to Layers

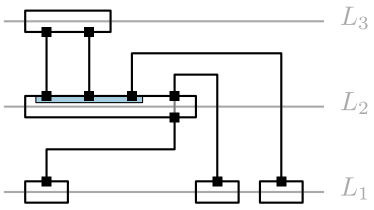
In this step, we seek an assignment of vertices to layers, such that all directed edges point upwards. We use a network simplex algorithm as described by Gansner et al. [GKNV93]. The algorithm is optimal in the sense that the sum of layers the edges span is minimized. With respect to the runtime of their algorithm, the authors state: "Although its time complexity has not been proven polynomial, in practice it takes few iterations and runs quickly."

7.3.3 Orienting Ports and Inserting Dummy Vertices

Consider the ports of a vertex. If a port group is of a type different than FREE, we assign all ports of this port group or a port group containing this port group to the specified vertex side, e.g., the bottom side. (Ignore for the moment the port groups of type LEFT and RIGHT. Below, we describe how to handle them.) If this



(a) We insert an extra layer $L_{2.5}$ to host a dummy vertex (solid red) as a turning point. All edges traversing a layer are subdivided by dummy vertices (hatched green).



(b) Each port of the vertex on L_2 is in a port group or in a port pairing. Thus, the two rightmost ports are placed on the top side, although they have incoming edges from below.

Figure 7.3: Example for the insertion of dummy vertices.

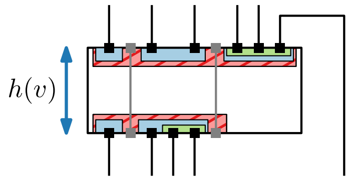
leads to contradicting assignments of the same port, then the input is inconsistent in assigning vertex sides to ports. We arbitrarily change vertex sides of affected port groups to obtain consistency. (Alternatively, one could reject such an instance.) We treat port pairings analogously. We assign ports that are in no port group to the top or the bottom side depending on whether they have an outgoing or incoming edge. If ports of a port group of type FREE remain unassigned, we make a majority decision for the top-level port group – if there are more outgoing than incoming edges, we set its ports to the top side; otherwise to the bottom side.

In any case, we may end up with ports being on the "wrong" side in terms of incident edges, e.g., a port on the top side has an incoming edge. To make such edges reach their other endpoints without running through the vertex rectangle, we introduce an extra layer directly above the layer at hand. On the extra layer, we then place a dummy vertex that serves as a "turning point" for these edges; see Figure 7.3. We refer to them as *turning dummy vertices.*

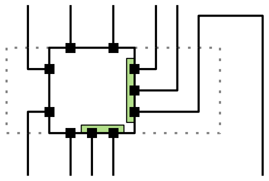
In contrast, KIELER [SSvH14] appends effectively, for each port that lies on the "wrong" side, a dummy port on the opposite side of the vertex rectangle, to the very right or left of the ports there. The edges are later routed around the vertex to this dummy port. Our new approach, therefore, provides somewhat greater flexibility in routing edges around vertices.

It remains to describe how to handle port groups of type LEFT and RIGHT. Note that our algorithm never assigns ports of a port group of type FREE to LEFT or RIGHT. However, the input data may contain port groups of these types.[15] Consider the port groups of type LEFT and RIGHT; see Figure 7.4 for this step. We assign their ports during the execution of the algorithm to the bottom or the top side of their vertices – again by a majority decision on their top-level port group. On the top and the bottom side, we introduce new top-level port groups with fixed order (hatched red in Figure 7.4a). They contain three port groups of free order (solid blue in Figure 7.4a) that contain everything on the left side, top/bottom side, and right side (in this order and each separated by two ports with a port pairing; gray in

[15] In our experiments, we do not have port groups of type LEFT or RIGHT. So, here, we suggest a general approach on how to handle this case, which we did not implement or test.



(a) Instead of ports on the left and the right side, we subdivide the top and bottom side into three port groups (solid blue) using a port group with fixed order (hatched red) and two port pairings.



(b) In a post-processing, we shrink a vertex to its middle part and re-route the edges entering a port on the left or right side of the vertex. The considered vertex has two port groups (solid green).

Figure 7.4: Construction to model ports on the left and the right side of a vertex.

Figure 7.4a). Later, we shrink each vertex v to its inner part and re-route the ends of the edges incident to ports in port groups of type Left and Right as L-shapes in the released area (interior of the dashed box in Figure 7.4b). Hence, we adjust $w(v)$ and $h(v)$ in the forehand accordingly.

After this step for handling port groups of type Left and Right, every port is assigned either to the top or the bottom side of its vertex.

As in the classical algorithms for layered graph drawing, we subdivide edges traversing a layer (which may also be an extra layer) by a new dummy vertex on each such layer. Hence, we have only edges connecting neighboring layers. As for all algorithms that rely on decomposing the edges, this phase runs in $\mathcal{O}(\lambda \cdot |E| + |P|)$ time, where λ is the number of layers. Note that $\lambda \in \mathcal{O}(|V|)$.

7.3.4 Reducing Crossings by Swapping Vertices and Ports

We employ the layer sweep algorithm using the well-known barycenter heuristic proposed by Sugiyama et al. [STT81]. However, we also have to take the ports and the port constraints into account. We suggest three ways to incorporate them.

Vertices: We first ignore ports. We arrange the vertices as follows. Since there may be many edges between the same pair of vertices, we compute the vertex barycenters weighted by edge multiplicities. After having arranged all vertices, we arrange the ports at each vertex to minimize edge crossings. Finally, we rearrange the ports according to port pairings and port groups by computing barycenters of the ports of each port group.

Ports: We use indices for the ports instead of the vertices and apply the barycenter heuristic to the ports. This may yield an invalid ordering with respect to port groups and vertices. Hence, we sort the vertices by the arithmetic mean of the port indices computed before. Within a vertex, we sort the port groups by the arithmetic mean of the indices of their ports. We recursively proceed in this way for port groups contained in port groups and finally for the ports.

MIXED: Vertices that do not have port pairings are kept as a whole, vertices with port pairings are decomposed into their ports. The idea is that, when sweeping up or down, the ports do not influence the ordering on the other side and can be handled in the end – unless they are paired. After each iteration, we force the ports from decomposed vertices to be neighbors by computing their barycenters, and we arrange the paired ports above each other. Finally, we arrange all ports that are not included in the ordering as in VERTICES.

In all cases, if a port group has a fixed order, we cannot re-permute its elements, but we take the order as described from left to right. We use random start permutations for vertices and ports. We execute this step r times for some constant r (in our experiments $r = 1$) and take the solution that causes the fewest crossings.

KIELER [SSvH14] also computes barycenters depending on the order of ports of the previous layer. Similar to PORTS they describe a *layer-total* approach and similar to MIXED they describe a *node-relative* approach. However, they compute barycenters only for vertices as a whole. We use barycenters of ports to recursively determine also an ordering of port groups.

It remains to describe how to handle a vertex v on a layer L_i that has edges in only one direction, say to the layer L_{i-1} below. In particular, this concerns turning dummy vertices of which we have many in our experiments. If we sweep upwards, we use v's neighbors on L_{i-1} to determine v's barycenter b_{v^-} in the usual way, which is

$$b_{v^-} = \frac{\sum_{u \in N(v) \cap L_{i-1}} \mathrm{pos}_{L_{i-1}}(u)}{|N(v) \cap L_{i-1}|} \,,$$

where $\mathrm{pos}_{L_{i-1}}(u)$ is the position of vertex u on layer L_{i-1}. However, if we sweep downwards, it is not clear how to arrange v relative to the other vertices on L_i since we cannot compute a barycenter using neighboring vertices on L_{i+1}.

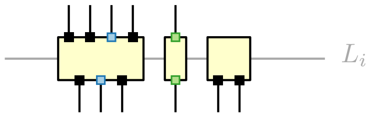
For these local sources and sinks, we investigate the following strategies.

PSEUDOBC: We compute and use a pseudo barycenter $b_{v^+}^{\text{pseudo}}$ being the current position of v on its layer L_i normalized by the number of vertices on L_{i+1}. More precisely, $b_{v^+}^{\text{pseudo}} = \mathrm{pos}_{L_i}(v) \cdot \frac{|L_{i+1}|}{|L_i|}$.

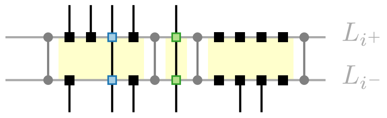
OPPOSITEBC: We compute and use a barycenter $b_{v^+}^{\text{opposite}}$ being the barycenter of v with respect to the opposite layer of L_i normalized by the number of vertices on L_{i+1}. More precisely, $b_{v^+}^{\text{opposite}} = b_{v^-} \cdot \frac{|L_{i+1}|}{|L_{i-1}|}$.

RELPOS: We do not compute any barycenter of v, but keep v at its current position within L_i. In other words, we remove v and all vertices without edges to L_{i+1} from L_i before computing the barycenters. Then, we sort the remaining vertices in the usual way according to their barycenters with respect to L_{i+1}. Finally, we re-insert v and all vertices without edges to L_{i+1} into the same positions they previously had on L_i.

This phase runs in $\mathcal{O}(r \cdot J \cdot \lambda \cdot |E|)$ time, where J is the number of (top-down or bottom-up) sweeps within one execution of the layer sweep algorithm.



(a) Three vertices with two port pairings on one layer before transforming them to two layers and ports only.



(b) Instead of vertices, we use ports on two layers. Port pairings and new ports for separating vertices are connected by a dummy edge. The rightmost vertex is "padded" to be wider using dummy ports.

Figure 7.5: Example of the transformation of vertices with ports on one layer to ports and edges on two layers; port pairings are indicated by color.

7.3.5 Determining Vertex Coordinates

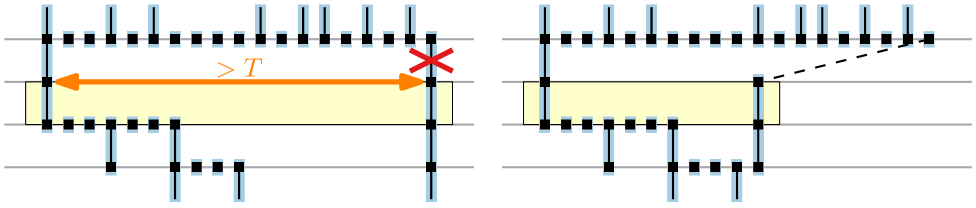
To position both vertices and ports, we decompose the vertices into ports and edges. An example is given in Figure 7.5. We duplicate each layer L_i (except for the extra layers introduced in Section 7.3.3) to an upper layer L_{i^+} and a lower layer L_{i^-}. For a vertex on layer L_i, we place all ports of the Top side in the previously computed order onto L_{i^+} and all ports of the Bottom side in the previously computed order onto L_{i^-}.

To separate the vertices from each other and to assign them a rectangular drawing area, we insert a path of length one with the one port on L_{i^-} and the other port on L_{i^+} at the beginning and the end of each layer and between every two consecutive vertices (gray with ports drawn as disks in Figure 7.5b). Moreover, we may insert dummy ports without edges within the designated area of a vertex, to increase the width of a vertex. This can be seen as "padding" the width of a vertex v via ports to obtain the desired minimum width $w(v)$. For each port pairing $\{p, p'\}$, where p is on L_{i^-} and p' is on L_{i^+}, we insert a dummy edge connecting p and p'. Similarly for each dummy vertex subdividing a long edge, we add a path of length 1 between L_{i^-} and L_{i^+}.

Observe that we do not have edge crossings between L_{i^-} and L_{i^+}. Therefore, using the algorithm by Brandes and Köpf [BK02] (see below), these edges end up as vertical line segments. This fulfills our requirement for vertices being rectangular and for ports of port pairings being vertically aligned.

Now we have a new graph G' with ports being assigned to layers, but without vertices and without port constraints. So, in the following, we consider the ports as vertices. This is precisely the situation as in the classical algorithms for layered graph drawing when determining coordinates of vertices. After the current coordinate assignment step, we will re-transform the drawing into our setting with vertices, ports, and edges.

The y-coordinate of a port is given by its layer. For assigning x-coordinates, we use the well-established linear-time algorithm by Brandes and Köpf [BK02]. (Alternative strategies were suggested, e.g., by Sander [San94]). The algorithm by Brandes Köpf heuristically tries to straighten long edges vertically and to balance the position of a port with respect to its upper and lower neighbors. It guarantees to preserve the



(a) finding a large gap within a vertex **(b)** breaking a block into two parts to narrow the gap

Figure 7.6: A wide vertex (yellow background color) arising when we employ the algorithm by Brandes and Köpf [BK02]. With an additional check, we detect large gaps between neighboring ports within a vertex and "break" the involved blocks. Blocks are highlighted by blue background color.

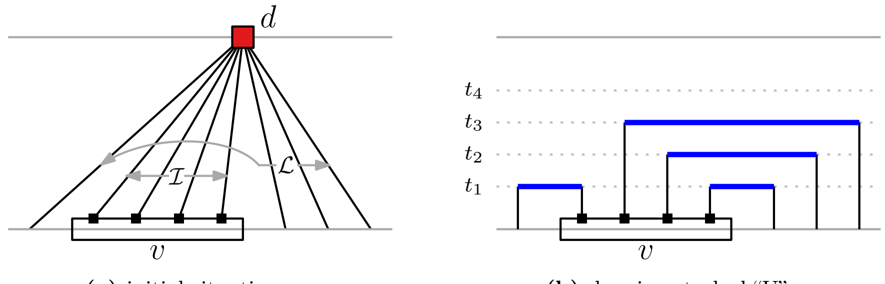
given port order on each layer and a minimum distance δ between consecutive ports. Moreover, it guarantees that uncrossed edges are drawn as vertical line segments, which is crucial for our application. Such a sequence of vertically stacked ports is called a *block*. Roughly speaking, the blocks are placed horizontally next to each other such that no two blocks overlap and the slack between the blocks is minimized.

We note that the original algorithm by Brandes and Köpf [BK02] contained two flaws that came up in our experiments. Subsequently, they were fixed [BWZ20].

Using the algorithm by Brandes and Köpf for ports instead of vertices has the drawback that vertices are drawn as relatively wide rectangles. This is because ports of the same vertex may be placed vertically above distant ports of the previous layer. To avoid these large gaps between ports of the same vertex, we extend the algorithm by Brandes and Köpf by the following check when placing the blocks. If two ports of two neighboring blocks are part of the same vertex and if the distance between these two ports is greater than a given threshold T (in our case 16 times the given minimum port distance), then we "break" one of the involved blocks into two blocks; see Figure 7.6. This means that one of the edges that has been a vertical edge within the block is not drawn as a vertical line segment. However, now the blocks are placed closer to each other effecting a smaller total width of the vertex.

It may happen that a large gap cannot be closed this way because we are not allowed to break port pairing edges. Therefore, we additionally do a post processing, where we forget about all blocks and structures within the algorithm by Brandes and Köpf and just consider each vertex individually. If large gaps remain, we push ports closer to each other where possible without breaking internal port pairings. Note that by avoiding wide vertices with both of these operations, we increase the number of bends in the resulting drawing since we lose vertical straight-line segments.

The algorithm by Brandes and Köpf runs in time linear in the number of ports and edges. Our modification breaks each block at most λ times, where λ is the number of layers. Hence, this phase runs in $\mathcal{O}(\lambda(|E| + |P|))$ time.



(a) initial situation **(b)** drawing stacked "U"s

Figure 7.7: Drawing edges going through turning dummy vertices orthogonally.

7.3.6 Constructing the Drawing and Routing the Edges Orthogonally

First, we obtain vertices drawn as rectangles from (dummy) ports and edges by reversing the transformation described in Section 7.3.5.

Then, we obtain edges drawn as polylines by transforming the dummy vertices inserted in Section 7.3.3 into bend points of their edges. We re-draw vertices with ports on the left or right side by shrinking the width of the vertex and extending the incident edges within the released area. For horizontal port pairings, we increase the height of a vertex and re-sort the ports on the left and the right side.

Finally, we draw the edges orthogonally using the approximation algorithm for coloring bidirectional interval graphs from Chapter 6. There, we do the theoretical description and the analysis of the approximation algorithm and the greedy algorithm it is based on. Here, we describe the practical implementation in more detail, in particular how to handle the special cases excluded there for a smoother analysis and how to proceed with objects being specific to our method, for example, turning dummy vertices.

Let us first describe how to draw the edges going through a turning dummy vertex d (red in Figure 7.3). This step is depicted in Figure 7.7. Recall that for each vertex v, we have up to one turning dummy vertex on the next layer above (for edges going downwards) and up to one in the next layer below (for edges going upwards). Without loss of generality, let d be on the next layer above v. Observe that we have an even number of edge pieces being adjacent to d as they correspond to edges entering and leaving d. Let $\mathcal{I}$ be the set of edge pieces entering d, and let $\mathcal{L}$ be the set of those leaving d. Those in $\mathcal{I}$ are incident to ports $P_{\mathcal{I}}$ of v. Where possible, we sort the ports of $P_{\mathcal{I}}$ at v such that the order of $\mathcal{I}$ is, for both the edges passing v on the left and on the right, inverse to their corresponding edge pieces in $\mathcal{L}$. This can be done in $\mathcal{O}(\lambda|E|)$ time in total using Bucketsort. The resulting order allows us to draw the edges as two stacks of (upside-down) "U"s as in Figure 7.7b. We greedily use intermediate horizontal lines $t_1, t_2, \ldots$, which we call *tracks*, to place the horizontal segments. Since we need at most $\mathcal{O}(|E|)$ tracks between any two layers and have at most $\mathcal{O}(\lambda|E|)$ edge pieces, the runtime for this step is in $\mathcal{O}(\lambda|E|)$. The greedy procedure is optimal for an individual vertex, but may produce avoidable crossings between different vertices depending on the order in which we process the dummy turning vertices.

For all other edge pieces spanning a layer, it remains to draw them orthogonally. We do not need to consider vertical segments since they are already drawn in the orthogonal style. Consider the remaining (skewed) edge pieces. Their endpoints are ports of vertices and dummy vertices. Let P be the set of these ports. We first assume that the x-coordinates of the ports on the two layers are all different. Below, we treat the general case.

Each port $u \in P$ has its x-coordinate $x(u)$. For an edge piece uv, where u is on the layer below the layer of v, we can compute its *span* as $\mathrm{span}(uv) = [\min\{x(u), x(v)\}, \max\{x(u), x(v)\}]$. Note that every span corresponds to an interval, which means the set of edge pieces between each two layers corresponds to an arrangement of intervals on the x-axis. Since we assume that all edge pieces point upwards, we can distinguish the intervals into *right-going* if $x(u) < x(v)$ and *left-going* otherwise. We want to draw each edge piece uv as a sequence of three axis-aligned line segments: vertical, horizontal, vertical; starting at u and ending at v. For the horizontal segments, we use tracks, as we did for the "U"s. Our task is to assign the horizontal segment of each edge piece to a track such that no two horizontal segments intersect and no two edge pieces cross twice. The objective is to minimize the number of tracks. This is precisely the situation of computing a minimum-size proper coloring of the corresponding bidirectional interval representation that we describe in detail in Chapter 6.

In Section 6.2, we show that we have a 2-approximation algorithm for coloring bidirectional interval graphs and representations if we separate the intervals into left- and right-going and solve both cases independently and optimally.

Let us recap the corresponding greedy algorithm for directional interval graphs. (Instead of a coloring, we directly speak of an assignment of intervals to tracks.) We assume, without loss of generality, that all intervals are left-going as depicted in Theorem 6.2; the case that the intervals are right-going is symmetric. We first sort the intervals by left endpoints and then we assign them to the lowest available track. A track is *available* for an interval $\mathrm{span}(uv)$ if there is no other interval $\mathrm{span}(wx)$ with $\mathrm{span}(uv) \cap \mathrm{span}(uv) \neq \emptyset$ assigned to it and all previous intervals that overlap (but do not contain) $\mathrm{span}(uv)$ are below this track. By Theorem 6.2, this greedy algorithm uses the minimum number of tracks and, over all layers, this can be accomplished in $\mathcal{O}(\lambda |E| \log |E|)$ time (see also Lemma 6.1).

Now let us consider the special case that for the left- and right-going edge pieces, there are ports with equal x-coordinates (connected by black dashed lines in Figure 7.9, top row). There, we must additionally make sure that their vertical segments do not intersect. To this end, we introduce an additional track $t^\star$ at the top to place an extra horizontal segment for all "problematic" cases, investing two additional bends; see Figure 7.9b and c. In Figure 7.9a (where the right endpoints have the same x-coordinate) no extra bends are needed because we place the left-going edge pieces below the right-going edge pieces.

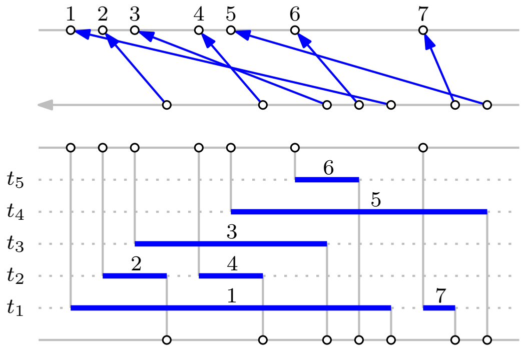


Figure 7.8: Drawing left-going edge pieces.

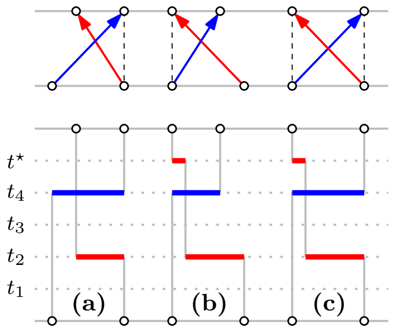


Figure 7.9: Equal x-coordinates.

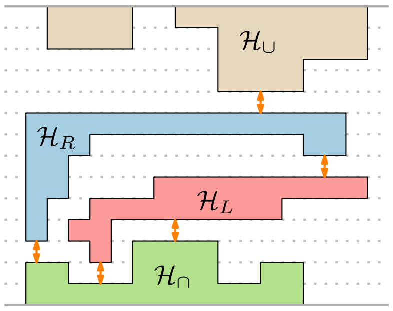


Figure 7.10: Moving the horizontal segments of the "U"s ($\mathcal{H}_\cup$), the right-going edge pieces ($\mathcal{H}_R$), the left-going edge pieces ($\mathcal{H}_L$), and the upside-down "U"s ($\mathcal{H}_\cap$) towards each other.

Finally, we move the horizontal segments $\mathcal{H}_R$ of right-going edge pieces simultaneously down until at least one of these segments, say a, is only one track above a horizontal segment, say b, (which in turn is in the group of horizontal segments $\mathcal{H}_L$ of left-going edge pieces) with $\operatorname{span}(a) \cap \operatorname{span}(b) \neq \emptyset$. We do the same for the "U"s ($\mathcal{H}_\cup$) on the top and the upside-down "U"s ($\mathcal{H}_\cap$) on the bottom. In other words, we move the blocks of horizontal segments towards each other until their contour lines would overlap if we would move by another track; see Figure 7.10. By Corollary 6.3, merging the left- and right-going edge pieces in this way, we obtain a 2-approximation in the number of tracks. We additionally have "U"s and upside-down "U"s. Moving them down and up should in practice rarely be beneficial, however, it is not clear how they affect the approximation factor precisely – thus, all we can guarantee is that this procedure is a 4-approximation with respect to the minimum number of tracks.

It remains to analyze the running time of this step. Between each two layers, we can merge all contour points into a list in $\mathcal{O}(|E|)$ time and then use a sweep-line approach to determine the distances between the contour lines between each two points of the list – again in $\mathcal{O}(|E|)$ time. So over all layers, this step can be performed in $\mathcal{O}(\lambda|E|)$ time.

Hence, the total running time of this phase is in $\mathcal{O}(\lambda|E| \log |E|)$.

7.4 Experimental Evaluation

For our experiments, we got access to 380 real cable plans of a large German machine manufacturing company (and another smaller data set; see Section 7.4.3). To obfuscate these plans and to have more data for our experiments, we generated 1140 pseudo cable plans from the real cable plans – three from each real cable plan. For replicability, we have made all of our algorithms, data structures, and data described here publicly available on github [pra20a, pra20b] – except for the original (company-owned) plans.

7.4.1 Graphs Used in the Experiments

First, we discuss the structure of these cable plans and how we transformed them to the format that is expected by our algorithm.

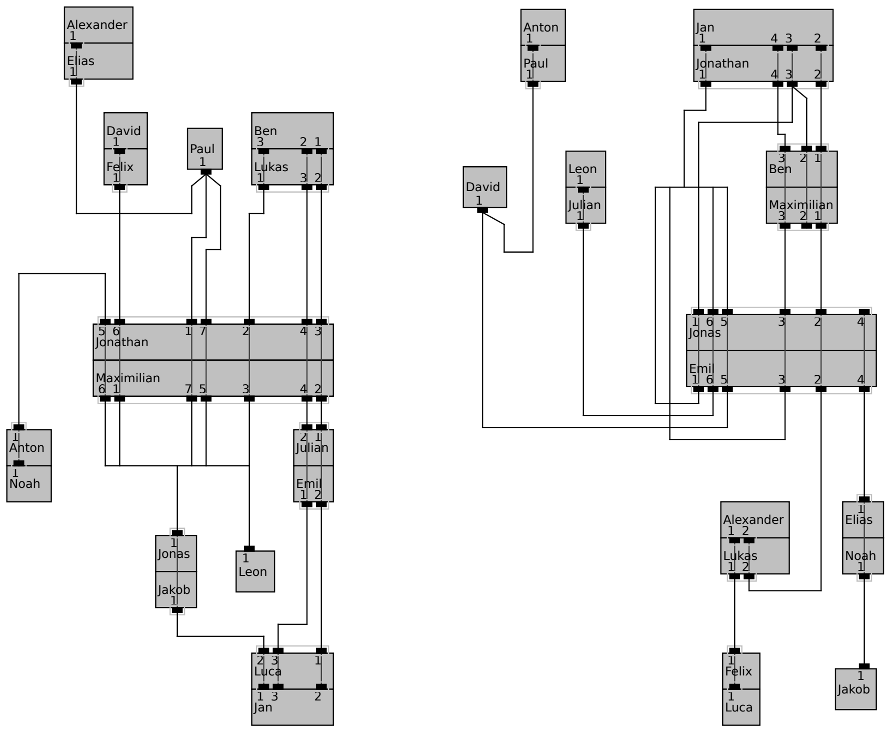
A cable plan has vertices with ports and vertex groups that comprise multiple vertices. Moreover, there can be edges connecting two or more ports (that is, hyperedges) and a port can be incident to an arbitrary number of edges. In a vertex group, there are port pairings between two vertices and these vertices should be drawn as touching rectangles. In our model, we do not have vertex groups and port pairings between different vertices. Instead, we model a vertex group as a single vertex with (internal) port pairings and a port group for the ports of each vertex. Moreover, we split ports of degree d into d separate ports and enforce that they are drawn next to each other and on the same side of the vertex by an (unordered) port group. We replace each hyperedge by a dummy vertex having an edge to each of the ports of the hyperedge. We neither have ports on the left nor on the right side of a vertex.

7.4.2 Generating a Large Pseudo Data Set from Original Data

Now, we describe briefly how we generated the pseudo cable plans. This can be seen as a method to extend and disguise a set of real-world graphs. A drawing of an original cable plan and a derived pseudo cable plan is depicted in Figure 7.11. In Section 7.7, we show larger examples of drawings of original cable plans and pseudo cable plans.

We generate a pseudo plan by removing and inserting elements from/to an original plan. Elements of the plans are the vertex groups, vertices, ports, port pairings, and edges. As a requirement, we had to replace or remove at least a q-fraction of the original elements (in our case this value was $q = .05$). We proceed in the following three phases.

1. We determine target values for most elements of the graph (number of vertex groups, vertices, ports, port pairings) and more specific parameters (distribution of edge–port incidences, arithmetic mean of parallel edges per edge, number of self loops, distribution of ports per edge, distribution of edges per port). We pick each target value randomly using a normal distribution,



(a) anonymized original cable plan

(b) artificial cable plan generated from the plan in ((a))

Figure 7.11: Example of an artificial cable plan generated from an original cable plan. Port groups are indicated by gray boxes and port pairings by line segments inside a vertex.

where the mean is this value in the original plan and the standard deviation is the standard deviation of this value across all graphs of the original data set divided by the number of plans in the original data set times a constant.

2. We remove a q-fraction of the original elements uniformly at random in the following order: vertex groups (incl. contained vertices and incident edges), vertices (incl. ports and incident edges), port pairings (incl. ports and incident edges), ports (incl. incident edges), and edges.

3. In the same order, we add as many new elements as needed to reach the respective target values. For the insertion of edges, we are a bit more careful. In case the graph became disconnected during the deletion phase, we first reconnect the graph by connecting different components. Then, we insert the remaining edges according to the distributions of edge–port incidences while trying to reduce the gaps between the target value and the current value for parallel edges per edge and for the number of self loops. Parallel edges have

the same terminal vertices but not necessarily the same terminal ports. We mostly use ports that do not have edges (they are new or their edges were removed or they had no edges initially) and assign for each one the number of edges it should get in the end. This gives us a set of candidate ports.

Next, we iteratively add a (hyper)edge e connecting d ports. In each iteration, we pick c sets of d ports from our set of candidate ports uniformly at random – each set is a candidate for the end points of the new edge. We choose the set where we approach the aforementioned target values the best if we would add the corresponding edge to the current graph. We used $c = 1000$, which means we took one out of 1000 randomly generated edge candidates.

Our generated pseudo cable plans are good if they are similar to and have similar characteristics as the original cable plans, and if the corresponding original cable plans cannot easily be reconstructed from the pseudo cable plans.

For our purposes, we can compare the results of the experiments using the original data set and the generated data set or we can compute explicit graph characterization parameters. The numbers of vertices, ports, edges, etc. are similar by using the target values. For example, the arithmetic mean (median) of the number of vertices in the original data set is 106.21 (106), while it is 106.15 (105.5) in the generated data set. The arithmetic mean (median) across the arithmetic means of parallel edges per edge in the original data set is 1.590 (1.429), while it is 1.491 (1.401) in the generated data set.

Some characteristic parameters where we did not have target values exhibit at least some similarities, which indicates a similar structure of the graphs of both sets. For example, the arithmetic mean (median) of the diameters across the largest components of all graphs in the original data set is 9.508 (10), while it is 8.731 (9) in the generated data set.

7.4.3 Experimental Setup

Our experiments were run in Java on an Intel Core i7 notebook with 8 cores (used in parallel) and 24 GB RAM under Linux and took about 3 hours.

We note that we have another smaller data set of 192 real cable plans where the vertex labels are common German male given names. We, therefore, call this data set *readable data set* and the previously described data set *large data set*. From the readable data set, we have generated pseudo cable plans as well. As it turned out, the statistical results for both data sets are very similar. This supports the stability of our results.

Due to the similarity of the results, we decided to detail only the results of the large data set in the description of our experiments. However, we present drawings of both data sets in Section 7.7, and the generated pseudo plans of both data sets are available in the git repository [pra20b].

Our experiments consist of two parts. In the first part, we compare the methods for orienting undirected edges, which we describe in Section 7.3.1 in more detail. In the second part, we compare the methods for reducing the number of crossings, which we describe in Section 7.3.4 in more detail. In the second part, we additionally compare these methods with the drawings generated by KIELER.

7.4.4 Orienting Undirected Edges

For each graph of the 1135 graphs and each of the variants FD, BFS, RAND, we oriented the edges and executed the algorithm ten times using the variant PORTS in the crossing reduction phase. For FD, we used only one execution of the force-directed algorithm (so $k = 1$) to make it better comparable to the other methods. We recorded

- the number n_{cr} of crossings in the final drawing,
- the number n_{bp} of bends created when executing the algorithm,
- the width, height, total area, and aspect ratio of the bounding box of the drawing, and
- the time to orient the edges and run the algorithm.

For each graph and each criterion, we took the best of the ten results for each method and then we normalized this best result by the best value of RAND. The arithmetic means (μ) of these values are listed in Table 7.1. The winner percentage β measures how often a specific method achieved the best objective value (usually the smallest, but for the aspect ratio (w:h) the one closest to 1). Ties are not broken, so over the three methods, the β-values add up to more than 100. We relate the normalized values of n_{cr} and n_{bp} to the number of vertices; see Figure 7.12 for the original plans and Figure 7.13 for the generated plans.

Table 7.1: Comparison of the methods for orienting the edges. The mean μ is relative to RAND (standard deviation in the range $[.1, .3]$); β measures (in %) how often a method provides the best result ($\sum \beta > 100$ possible due to ties).

	original cable plans						generated artificial cable plans					
	FD		BFS		RAND		FD		BFS		RAND	
	μ	β	μ	β	μ	β	μ	β	μ	β	μ	β
n_{cr}	.57	**78**	.64	31	1	11	.66	**87**	.76	25	1	11
n_{bp}	.94	**68**	.96	33	1	15	.96	**75**	.99	22	1	17
width	.56	**92**	.75	12	1	2	.64	**93**	.80	9	1	2
height	1.80	3	1.42	4	1	**97**	1.69	1	1.37	4	1	**98**
area	.98	**54**	1.04	18	1	33	1.06	27	1.06	26	1	**50**
w:h	.49	**85**	.65	17	1	3	.56	**86**	.73	13	1	5
time	1.10	3	.81	**97**	1	11	1.17	2	.87	**90**	1	19

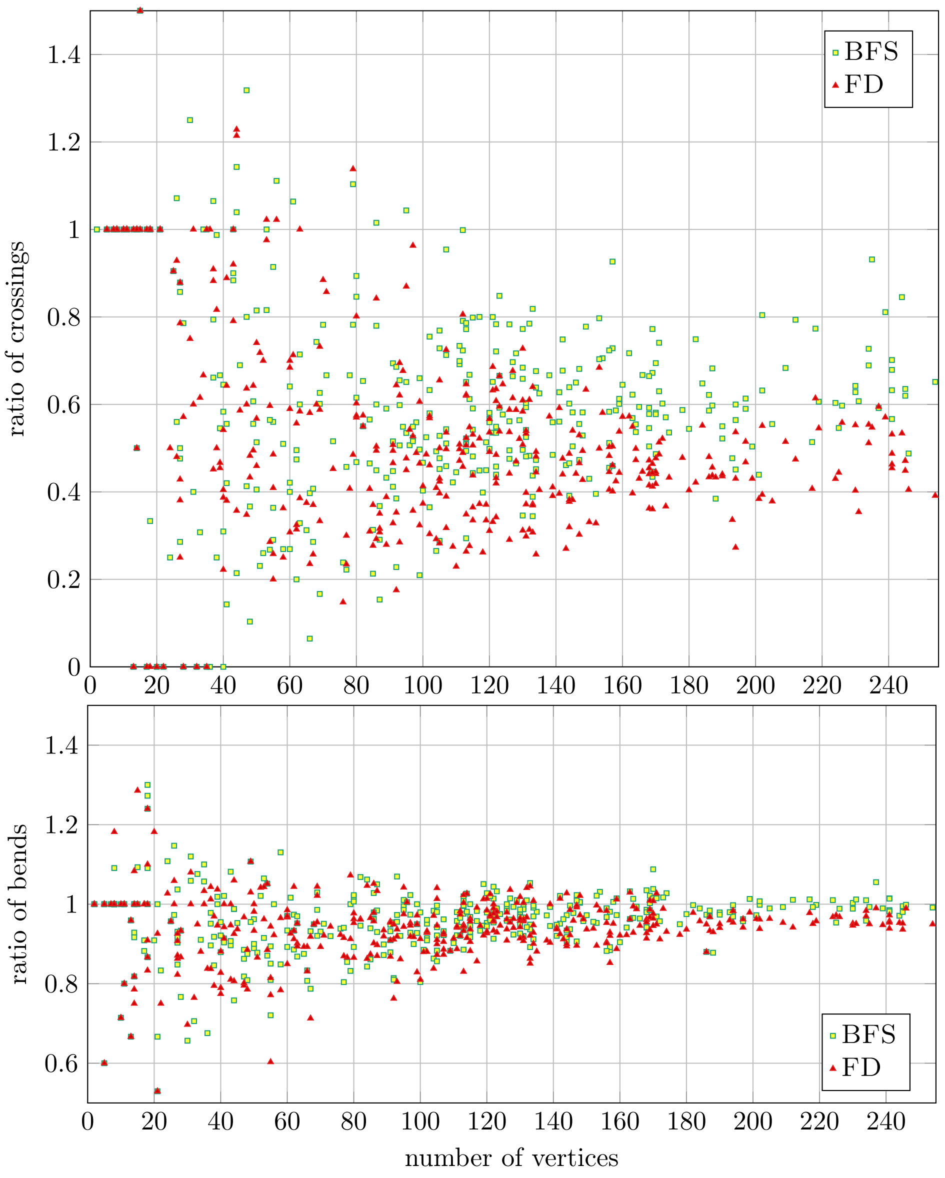
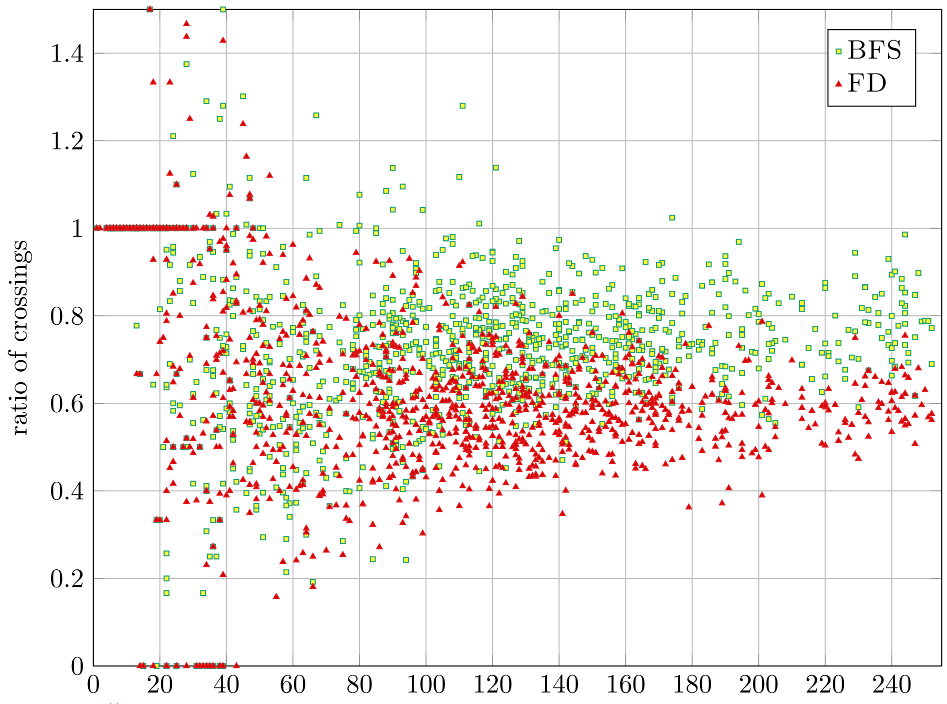


Figure 7.12: Comparison of the edge-orientation methods FD and BFS relative to Rand. In each color, each dot represents one of the 380 original plans.



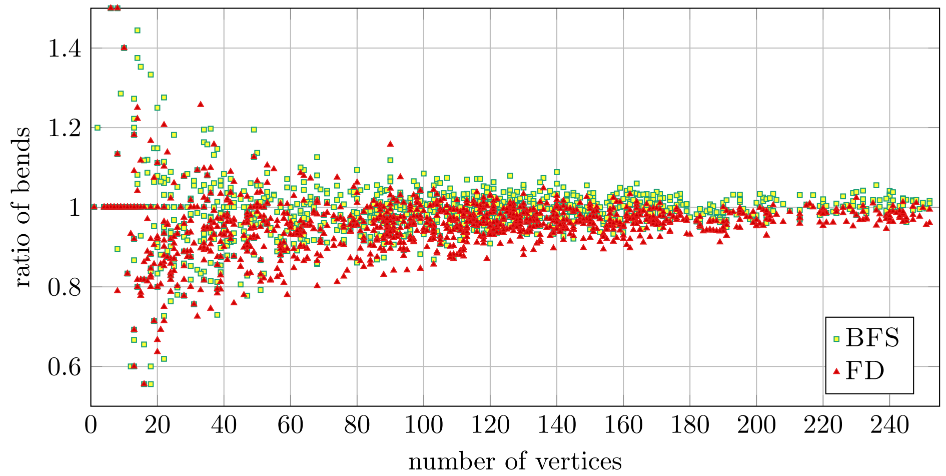


Figure 7.13: Comparison of the edge-orientation methods FD and BFS relative to RAND. In each color, each dot represents one of the 1140 generated plans.

7.4.5 Crossing Reduction

We used the same settings as when we compared the methods for orienting the edges, but here we exclusively used FD for orienting the edges. We compared the methods Vertices, Mixed, and Ports, and the methods PseudoBC, OppositeBC, and RelPos each with a single run of the crossing reduction phase. Kieler joined the comparison as the baseline method to which we relate our results.

The variant Kieler uses instead of our algorithm the algorithm ElkLayered in eclipse.elk (formerly known as: KLayered in KIELER) [elk20]. As our algorithm, ElkLayered does Sugiyama-based layered drawing using ports at vertices. ElkLayered, however, expects a directed graph as input and its port constraints are less powerful. ElkLayered offers free placement of the ports around a vertex, fixed side at a vertex, fixed order around a vertex, and fixed position at a vertex. After orienting the given undirected graph, we used this algorithm as a black box when we set the port constraints to the most flexible value for each vertex. So, for vertices having multiple port groups or port pairings, we set the order of ports to be fixed, while we allow free port placement for all other vertices. As both algorithms expect different input, use different subroutines and ElkLayered uses more additional steps for producing aesthetic drawings, this comparison should be treated with caution.

For our results, see Table 7.2 and Figures 7.14 and 7.15.

7.5 Discussion

In this section, we discuss the findings of our experiments in regards to the following aspects.

7.5.1 Methods for Orienting Undirected Edges

FD almost always yields orientations of the undirected graphs that lead to drawings with fewer crossings than the orientations obtained from BFS and Rand. The gap between FD and BFS is minor, whereas the gap between both FD and BFS to Rand is large. Regarding the bend points, there is a rather negligible advantage for FD and BFS. Comparing the drawing area, FD and BFS are similar, but FD achieves a better aspect ratio. Although Rand performs rather poorly for most criteria, it often uses the smallest drawing area. The savings in the total area by Rand can be attributed almost exclusively to a small height, which corresponds to fewer layers.

The layer assignment procedure uses more layers if we have longer paths of directed edges. FD rather straightens a path between two (distant) vertices requiring then more layers, while Rand rather orients some of the edges of this path up and some down, yielding shorter chains of directed edges. So, Rand has more vertices per layer, which also explains the worse width and aspect ratio. We suspect that this large width might partially be explained by the use of the algorithm by Brandes and Köpf [BK02] in the coordinate assignment phase. In this phase, many edges are drawn vertically. After the crossing minimization phase, we would expect that the

Table 7.2: Comparison of the methods for crossing reduction. The mean μ is relative to KIELER; the standard deviation is in the range $[.1, .7]$; β is as in Table 7.1.

	Original cable plans								Generated artificial cable plans							
	VTCS.		MIXED		PORTS		KIEL.		VTCS.		MIXED		PORTS		KIEL.	
	μ	β	μ	β	μ	β	μ	β	μ	β	μ	β	μ	β	μ	β
	PSEUDOBC															
n_{cr}	1.57	12	1.54	15	1.53	16	1	**83**	1.50	11	1.52	11	1.51	12	1	**94**
n_{bp}	1.05	12	1.03	25	1.03	19	1	**64**	1.06	11	1.05	16	1.04	17	1	**79**
width	1.06	17	1.06	17	1.05	16	1	**54**	1.12	13	1.13	11	1.12	13	1	**69**
height	1.36	6	1.36	4	1.36	5	1	**91**	1.42	1	1.43	1	1.42	1	1	**98**
area	1.43	6	1.42	7	1.42	6	1	**85**	1.60	2	1.61	2	1.61	2	1	**97**
w:h	.91	**30**	.91	27	.91	27	1	18	.91	**29**	.91	28	.91	29	1	16
time	1.09	50	1.25	14	1.31	9	1	**52**	1.45	13	1.80	8	1.95	7	1	**92**
	OPPOSITEBC															
n_{cr}	1.07	35	1.11	22	1.03	**38**	1	32	1.12	36	1.22	17	1.15	28	1	**45**
n_{bp}	1.04	12	1.02	26	1.03	22	1	**61**	1.05	12	1.04	18	1.04	17	1	**75**
width	1.13	17	1.13	14	1.14	15	1	**59**	1.23	6	1.25	6	1.24	6	1	**89**
height	1.31	6	1.32	6	1.31	4	1	**90**	1.38	1	1.38	1	1.38	1	1	**98**
area	1.48	6	1.50	7	1.49	4	1	**87**	1.72	2	1.73	2	1.72	2	1	**97**
w:h	.93	**33**	.93	23	.93	27	1	19	.95	29	.95	29	.95	**29**	1	15
time	1.61	23	1.96	11	1.81	13	1	**75**	2.30	10	2.77	7	2.83	6	1	**94**
	RELPOS															
n_{cr}	.82	23	.72	47	.70	**54**	1	9	.92	35	.90	41	.89	**42**	1	16
n_{bp}	1.04	13	1.03	20	1.02	26	1	**60**	1.06	10	1.04	18	1.04	18	1	**75**
width	1.11	13	1.08	19	1.08	17	1	**54**	1.21	6	1.20	8	1.21	6	1	**86**
height	1.29	5	1.29	5	1.29	5	1	**91**	1.38	1	1.36	1	1.37	1	1	**99**
area	1.43	7	1.39	9	1.40	7	1	**81**	1.67	2	1.65	2	1.66	2	1	**97**
w:h	.93	27	.93	25	.92	**31**	1	20	.94	26	.94	**31**	.94	30	1	15
time	1.07	48	1.20	12	1.24	11	1	**51**	1.43	13	1.76	9	1.86	8	1	**92**

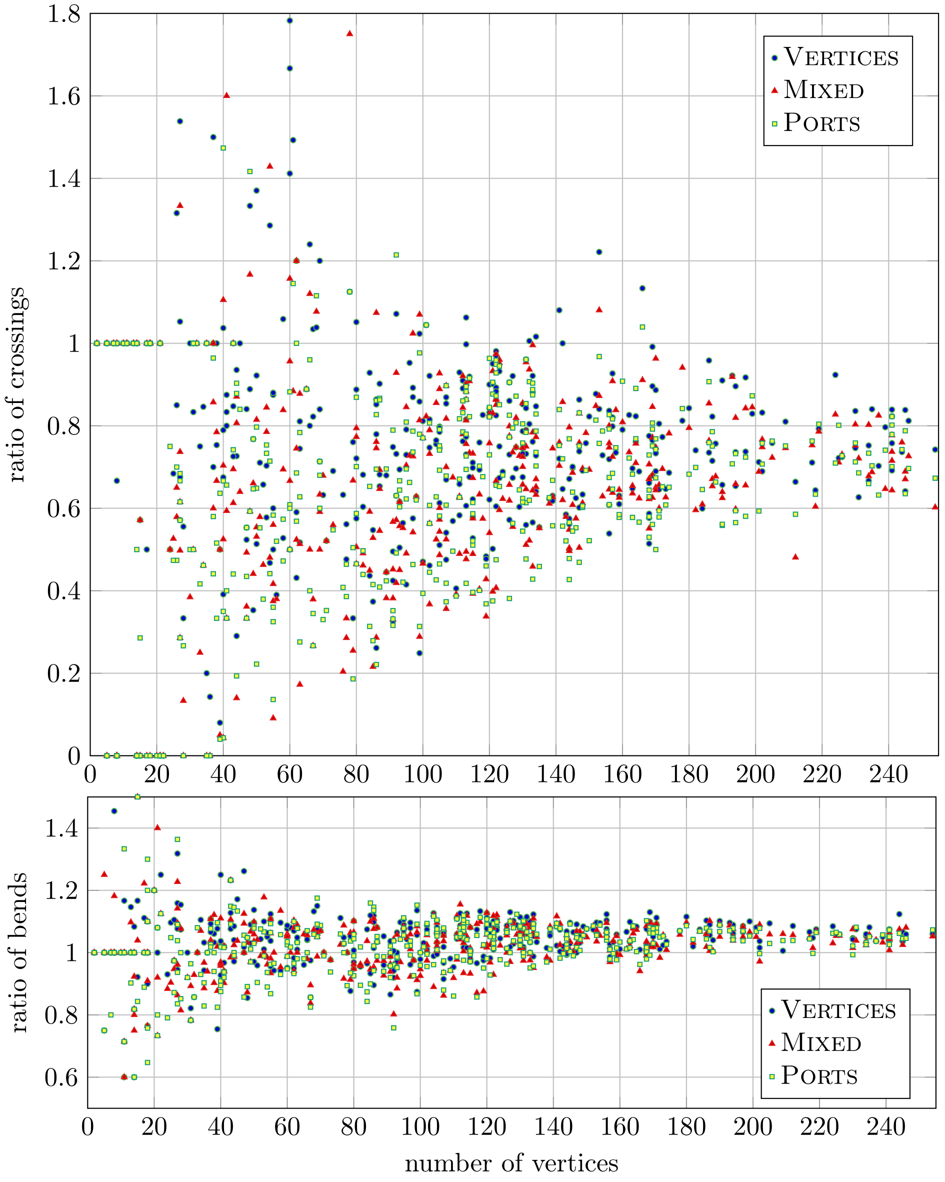


Figure 7.14: Comparison of the three crossing-reduction methods relative to Kieler. For handling local sources and sinks, we used RelPos. In each color, each dot represents one of the 380 original cable plans.

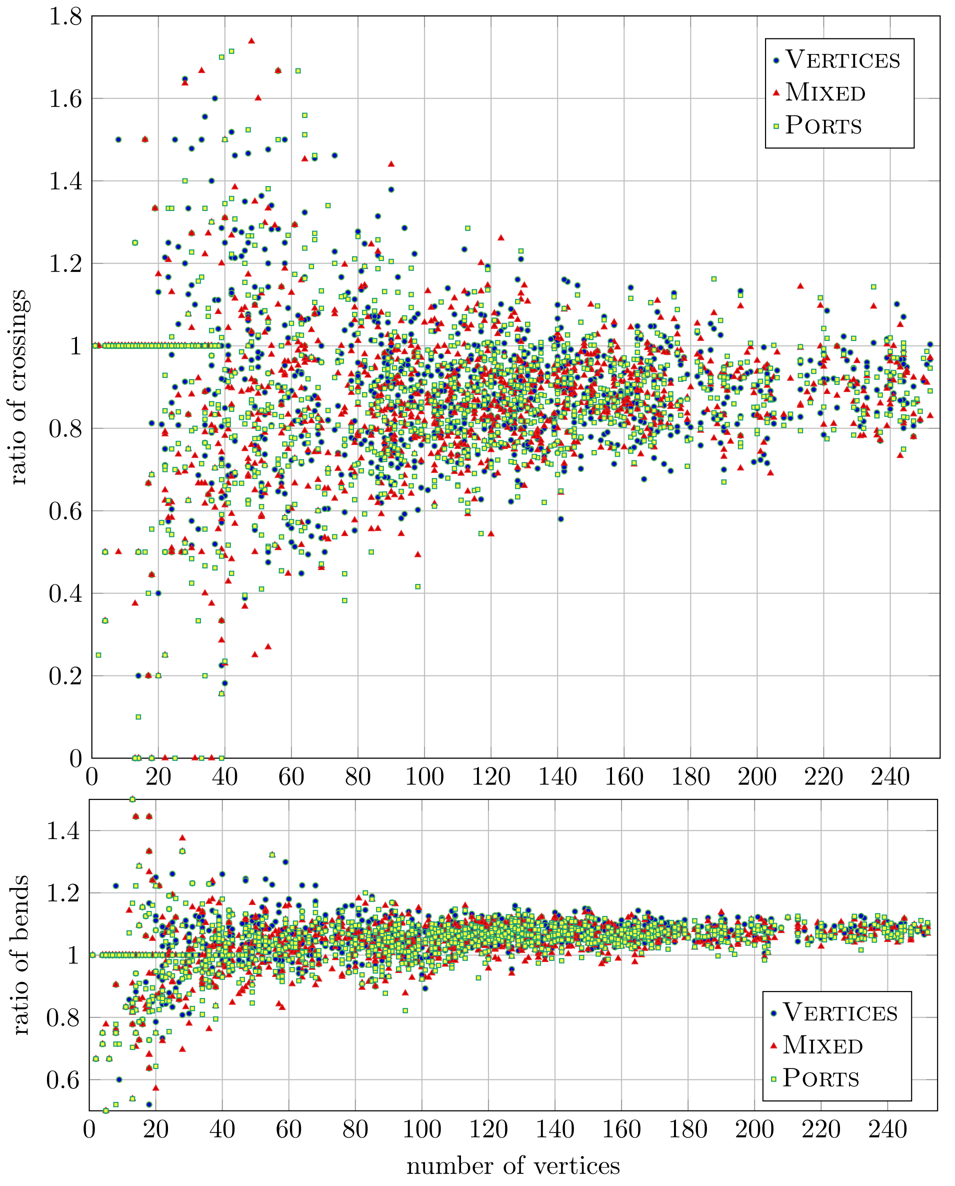


Figure 7.15: Comparison of the three crossing-reduction methods relative to Kieler. For handling local sources and sinks, we have used RelPos. In each color, each dot represents one of the 1140 generated cable plans.

vertices on the layers come close to the initial non-layered drawing of FD having short edges. When the edges between each two layers are longer for Rand, straightening them to a vertical line segment pushes vertices on the upper layer further apart from vertices on the lower layer.

Comparing the running times of the three variants, we note that using FD is about 10–20 % slower and using BFS is about 10-20 % faster than using Rand. We remark that these percentages refer to the running time of the whole algorithm, not just to the edge orienting phase. This explains why Rand is not necessarily the fastest variant; e.g., if Rand produces many dummy vertices and wider layers, the crossing reduction phase may take longer.

Summing up, we remark that it is worth using a more sophisticated method (FD or BFS) for orienting the undirected edges than just using a random assignment (Rand). The choice between FD and BFS depends on the user's preferences. FD tends to produce fewer crossings and a more balanced aspect ratio. BFS, in contrast, is (slightly) faster and conceptually simpler to understand and to implement. As our main goal is obtaining visually pleasant drawings, we recommend using FD for orienting edges if a (fast) force-directed graph drawing algorithm is available.

7.5.2 Methods for Crossing Reduction

We first consider the method for handling local sources and sinks in the layer-sweep algorithm. Then we analyze the methods for treating ports and vertices and compare them to Kieler.

Methods for Handling Local Sources and Sinks

Regarding the number of edge crossings, the rather simple approach RelPos outperforms PseudoBC and OppositeBC by far. The second-best method is clearly OppositeBC, whereas PseudoBC performs rather poorly. Regarding the number of bends and the drawing area, all three approaches behave quite similarly. RelPos and PseudoBC are about 50–70 % faster than OppositeBC, with a slight advantage for RelPos.

In our experiments, RelPos turned out to clearly be the best method or at least as good as the others, both in terms of simplicity and in terms of the criteria we measured. Therefore, we recommend RelPos, and in all remaining experiments, we use RelPos.

Methods for Treating Ports and Vertices

In terms of number of edge crossings, the methods Ports and Mixed achieve similar results; both clearly beat the method Vertices. This is in line with our expectation that incorporating distinct port orderings during the whole crossing reduction procedure helps to avoid edge crossings, which crucially depend on the precise order of ports. However, incorporating all ports (Ports) instead of only ports at vertices with port pairings (Mixed) does not seem to provide much of an additional benefit. (Recall that Mixed is not a generalization of Ports, but rather a generalization of VERTICES as the whole accounting is vertex-based instead of port-based.)

Regarding the number of bends and the drawing area, all variants perform similarly well. As expected, using VERTICES is faster than using MIXED, which in turn is faster than PORTS. Using PORTS, the total running time increases by about 10–40 % with respect to VERTICES.

Since we deem a small number of crossings the most important quality measure, we recommend using MIXED or PORTS, which we consider both equally well suited for our application.

7.5.3 Comparison to Kieler

Regarding the number of edge crossings, our new methods outperform the existing algorithm, which has not been designed for these specific port constraints. For the original cable plans, MIXED and PORTS use about 30 % fewer crossings and VERTICES still achieves about 20 % fewer crossings.

The number of bends is about the same for all variants of our algorithm and KIELER – we use on average at most 6 % more bends. We remark that this highly depends on the width of our vertex rectangles. Remember that we have adjusted the algorithm by Brandes and Köpf [BK02] to handle ports instead of vertices and to limit the distance of ports within the same vertex. Allowing an arbitrary placement also for ports of the same vertex leads to fewer bends, but also produces drawings with overwide vertices. In an earlier version of our algorithm [WZBW20], we did not limit the distance between ports within the same vertex. Additionally, our implementations differed in some other minor aspects. This resulted in our variants using much fewer bends than KIELER. Now we have made a design choice to avoid large gaps between ports within a vertex. (Recall that we break vertical alignments if gaps are larger than 16 times the minimum port distance.) We observed that due to this choice the drawings are sufficiently compact and vertex rectangles have an appealing aspect ratio. Moreover, we use roughly as many bends as KIELER does.

The drawings generated by our new algorithm use an about 40 % (original plans) and 65 % (artificial plans) larger area than the ones generated by KIELER. The main difference comes from a greater height, which we get from more horizontal tracks being used for the orthogonal edge routing and for integrating the intermediate layers that we use for turning dummy vertices. However, as the drawings generated by KIELER tend to be wider than high, using a greater height leads to a better aspect ratio for our variants (better in the sense of being closer to 1, i.e., the bounding box being more square-like).

Also with respect to the running time, KIELER produces its drawing on average a little faster than our algorithm. On the original plans, our variants need on average almost the same time (VERTICES) or about 25 % more time (PORTS). This gap is larger for the generated artificial cable plans, but it still seems to be in the range from a factor of 1 to a factor of 2 compared to KIELER. In total numbers, VERTICES, MIXED, PORTS, and KIELER needed in average for the original plans 142 ms, 166 ms, 173 ms, and 127 ms, respectively. The maximum running time that we measured occurred in a cable plan with 354 vertices. It took 1.1 s, 1.3 s, 1.6 s, and 0.6 s, respectively.

In conclusion, we can say that there is no algorithm being superior in all considered aspects. Cognitive studies, however, have shown that a small number of crossings highly influences the readability of a graph drawing for a human user [PCA02, WPCM02]. Our industry partners gave us similar feedback when working with these cable plans. Therefore, we consider reducing the number of crossings by more than a fourth and almost a third to be more important than a slightly smaller drawing area (which is likely to be less readable) and a slightly faster running time (which has to be done only once). Therefore, we recommend using our new algorithm with the variants Ports or Mixed in combination with RelPos when working with generalized port constraints and – more specifically – when working with cable or circuit plans that are somehow similar to the ones in our experiments.

We remark that the application settings that Kieler is designed for is not the same as for our algorithm, which limits the meaningfulness of this comparison. Moreover, Kieler uses more intermediate steps and post-processing steps, e.g., for compactification, which partially explains the smaller drawing area. Kieler also has more additional functionalities and is the overall more mature and established library. Kieler also provided an excellent starting point for our research and helped us to quickly generate some initial layered cable and circuit plans for our industrial partners.

7.5.4 Generating Pseudo Cable Plans

We concede that the artificial plans that we generated are not perfect as they behave somewhat differently from the original plans for certain criteria. For instance, for the artificial plans, the relative advantage of Ports and Mixed compared to Vertices vanishes. Also, our variants perform worse compared to Kieler with respect to the number of edge crossings, drawing area, and running time. Nevertheless, the obfuscation allowed us to make somewhat realistic cable plans publicly available, so that others can validate our experiments in the future.

7.6 Concluding Remarks and Open Problems

Our generation procedure may also serve as an entry point for more research in generating pseudo data from original data. This approach can be applied in many domains (and has most probably been applied, in domains we are not aware of). Finding such connections and formalizing the theory behind our obfuscation procedure would be interesting.

We are currently in the process of integrating our algorithm into the software of our industrial partner. We hope to see whether the statistical improvement of our algorithm actually yields advantages in practice. We also hope for practically relevant feedback and problems, which we can theoretically formalize and integrate in our model and algorithm.

We have not yet investigated the usual tuning of parameters much, e.g., the number of repetitions for the crossing reduction phase (currently $r = 1$) or more repetitions of the whole procedure. Besides minor tuning, our algorithm still leaves room for more radical improvements in many spots. This regards mainly the crossing reduction phase, the node/port placement phase, and the edge routing phase.

We are also interested in more domains where we can apply the concept of layered graph drawing with generalized port constraints – both for directed and undirected graphs. Besides cable plans, applications may include circuit plans, IT network plans, UML diagrams, data-flow networks, knowledge graphs, containment hierarchies, and many more.

7.7 Cable Plan Drawings

In Figures 7.16 to 7.27, we provide drawings of six cable plans (three original plans and three pseudo plans). For each plan, there is a drawing generated by our algorithm using FD, Ports, and RelPos, and there is another drawing generated using Kieler. The drawings have been generated automatically in a run where each plan has been drawn ten times and the best drawing (with respect to the number of crossings) has been kept. Port pairings are indicated by line segments inside a vertex.

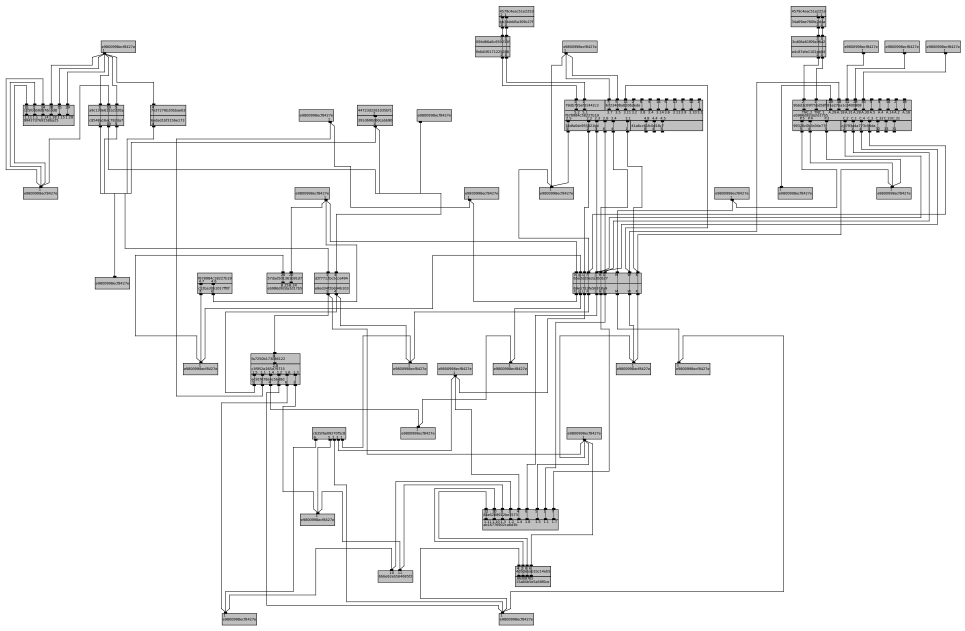


Figure 7.16: Original cable plan (anonymized) from the large data set with 69 vertices and 104 crossings drawn by our algorithm using FD, Ports, and RelPos.

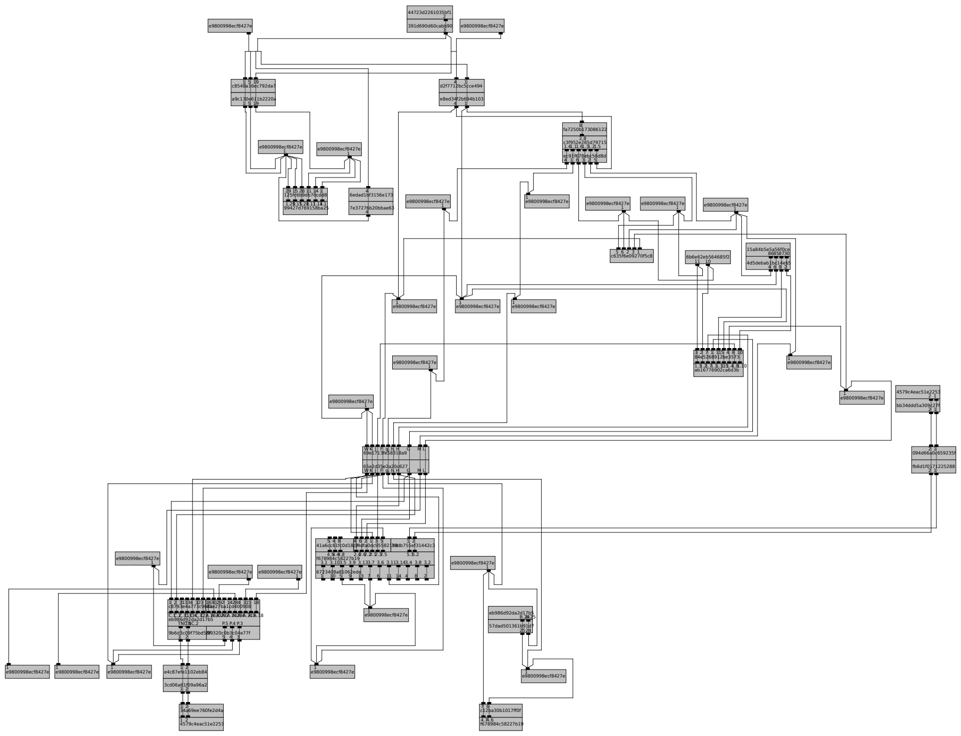


Figure 7.17: Original cable plan (anonymized) from the large data set with 69 vertices and 162 crossings drawn using KIELER.

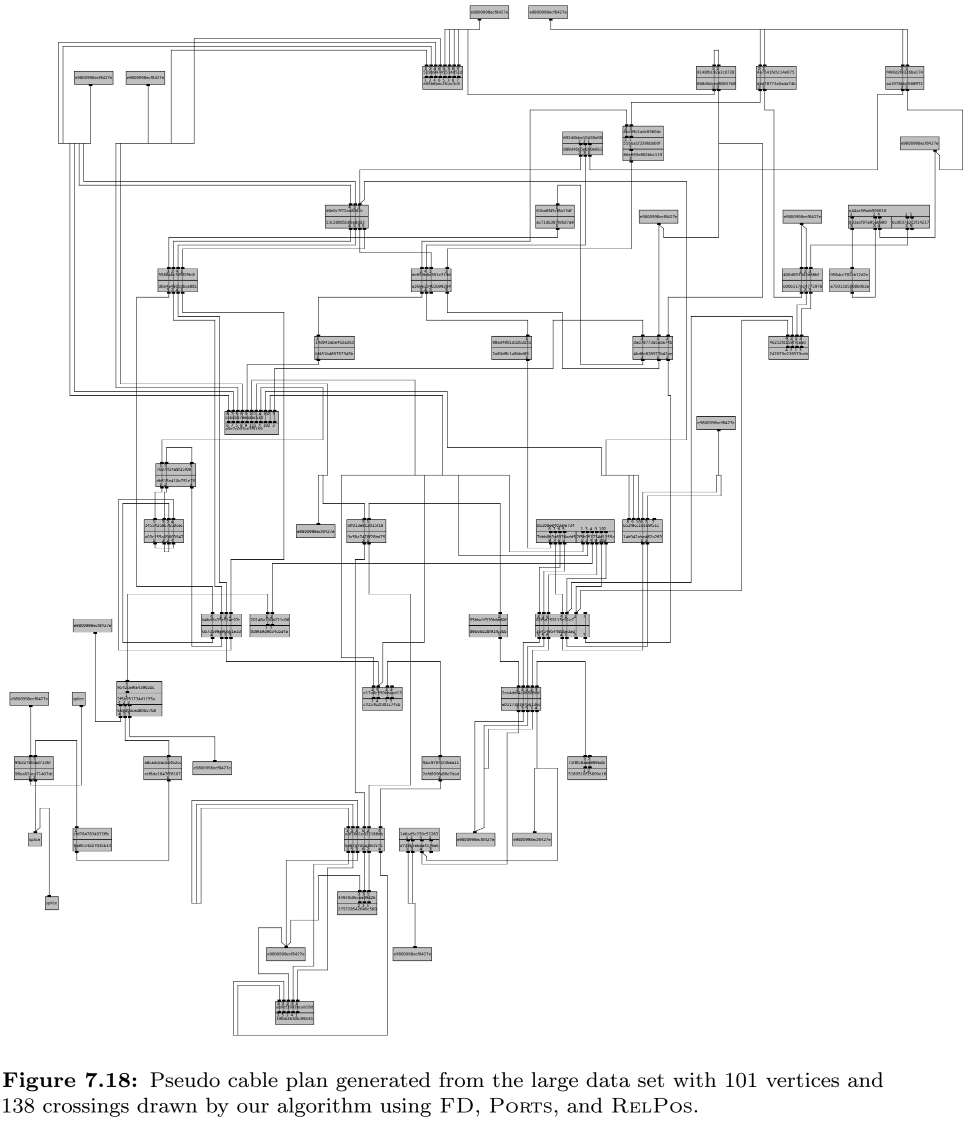


Figure 7.18: Pseudo cable plan generated from the large data set with 101 vertices and 138 crossings drawn by our algorithm using FD, Ports, and RelPos.

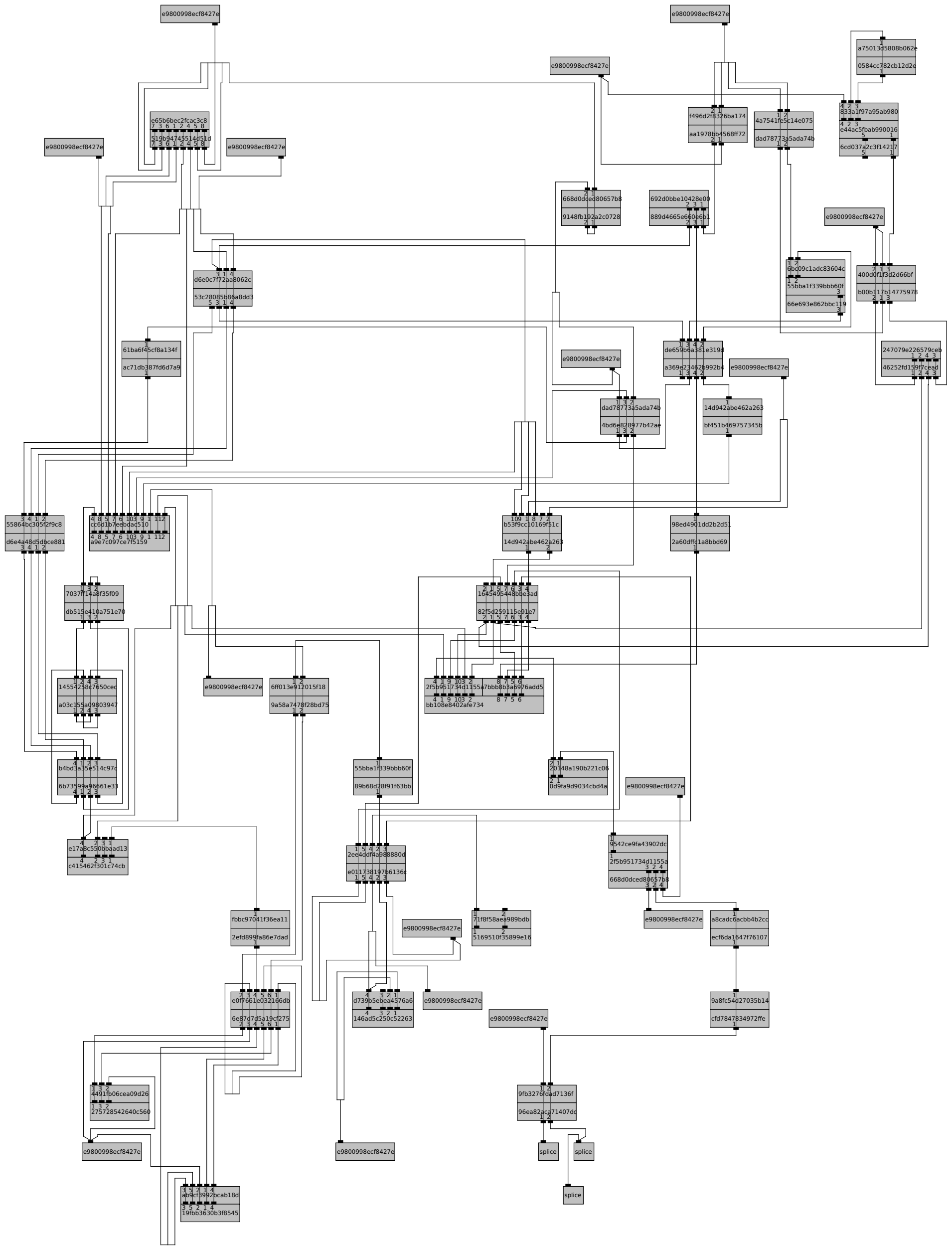


Figure 7.19: Pseudo cable plan generated from the large data set with 101 vertices and 149 crossings drawn using KIELER.

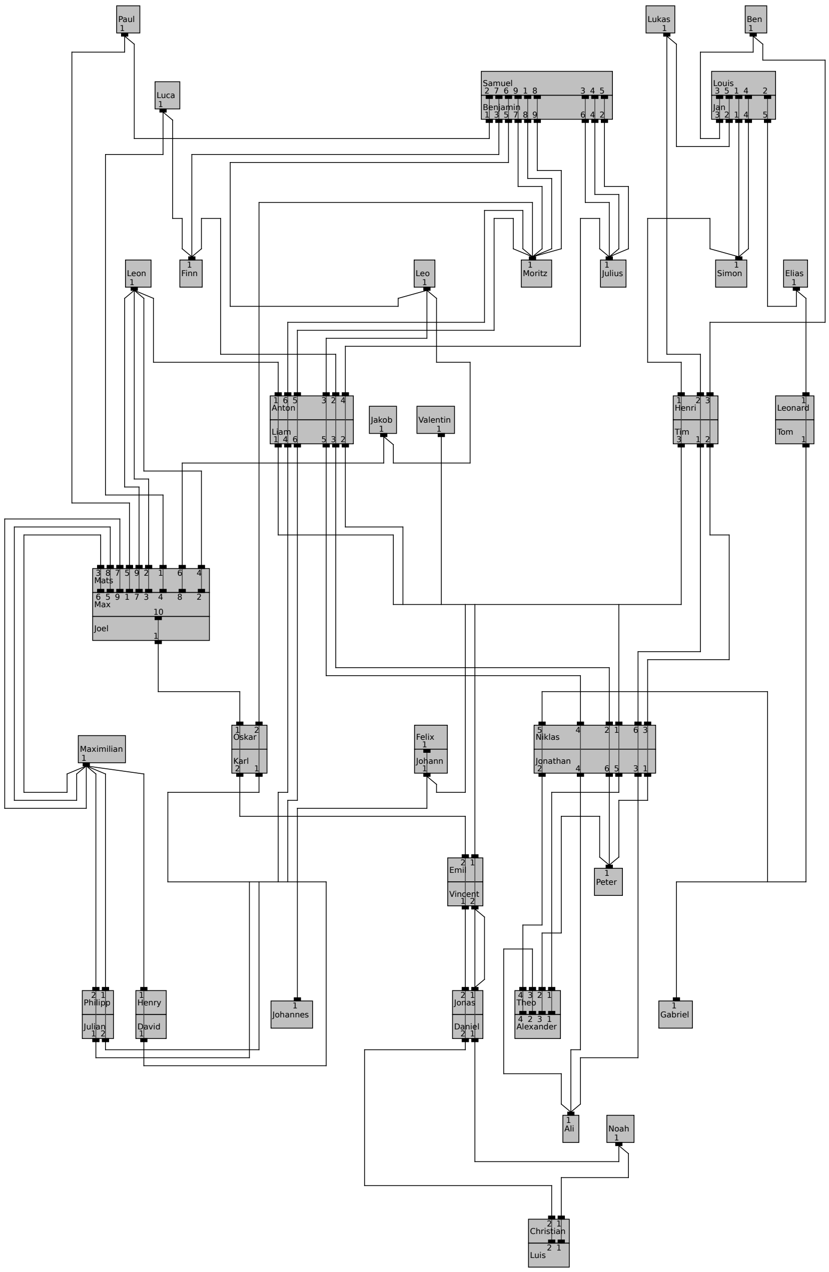


Figure 7.20: Original cable plan (anonymized) from the readable data set with 50 vertices and 52 crossings drawn by our algorithm using FD, Ports, and RelPos.

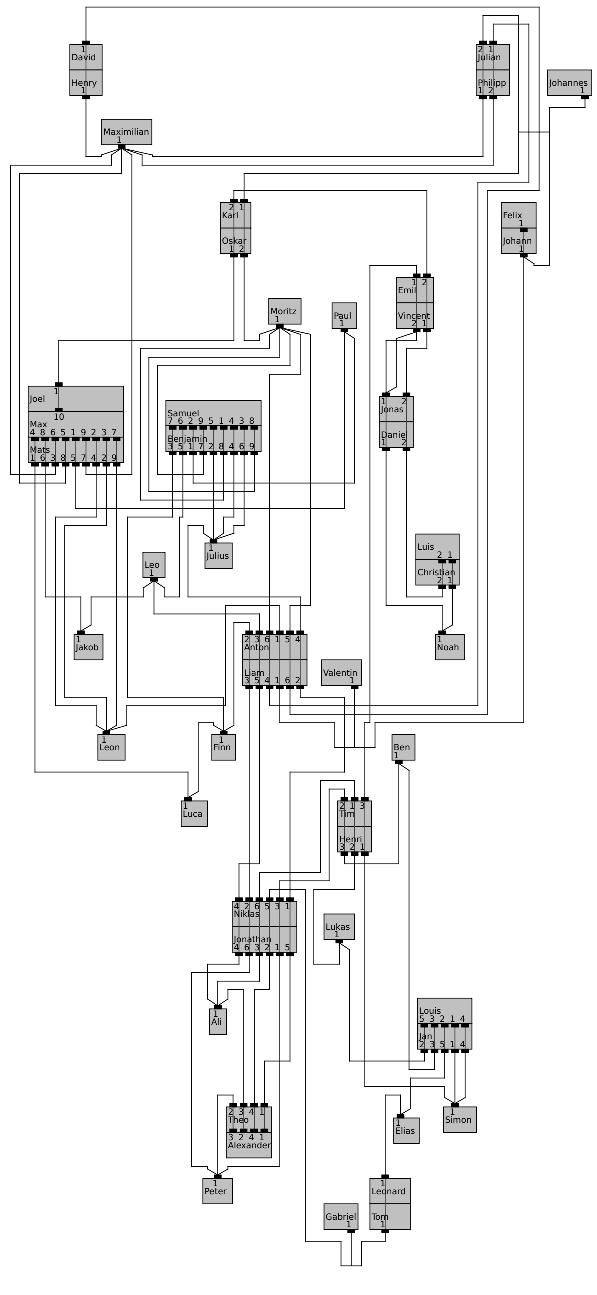


Figure 7.21: Original cable plan (anonymized) from the readable data set with 50 vertices and 71 crossings drawn using Kieler.

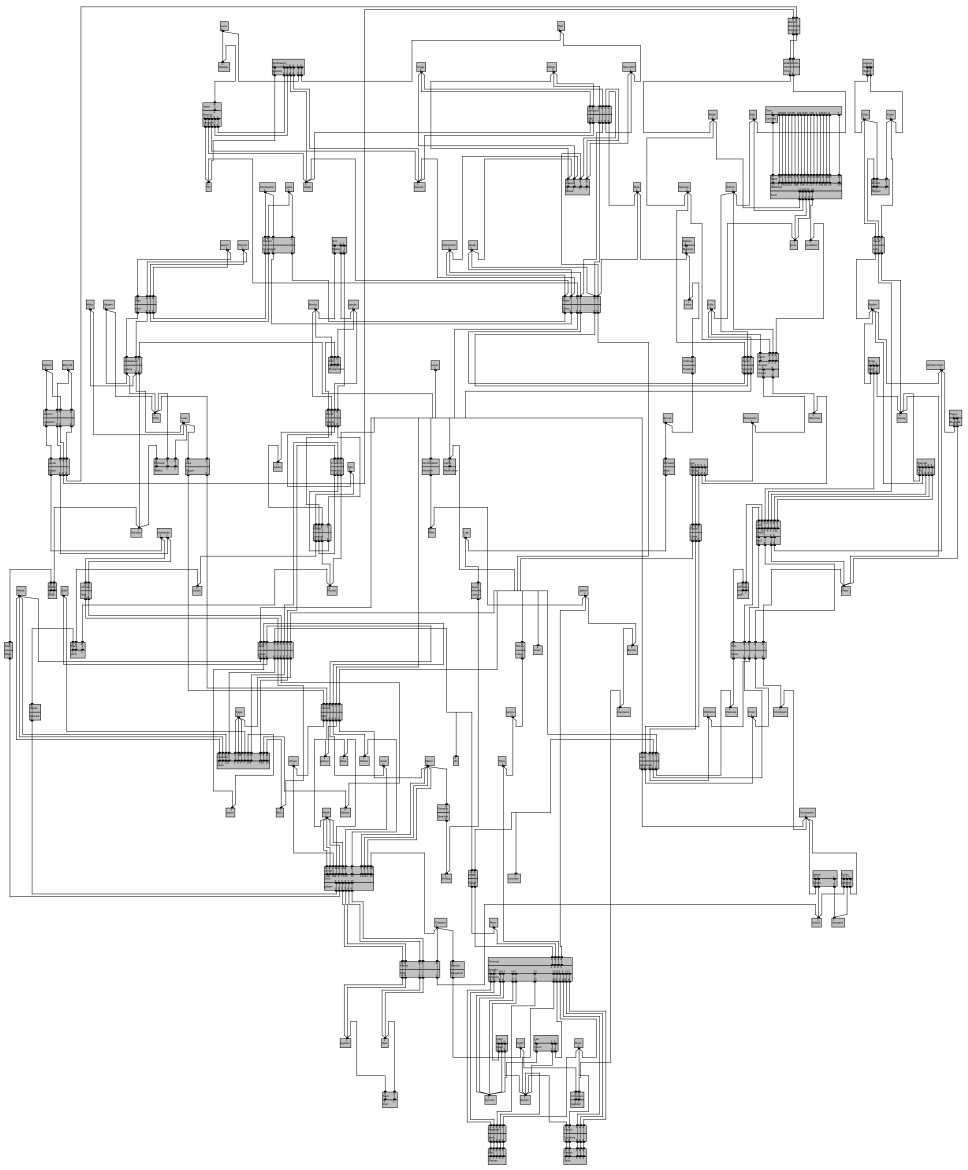


Figure 7.22: Original cable plan (anonymized) from the readable data set with 225 vertices and 391 crossings drawn by our algorithm using FD, Ports, and RelPos.

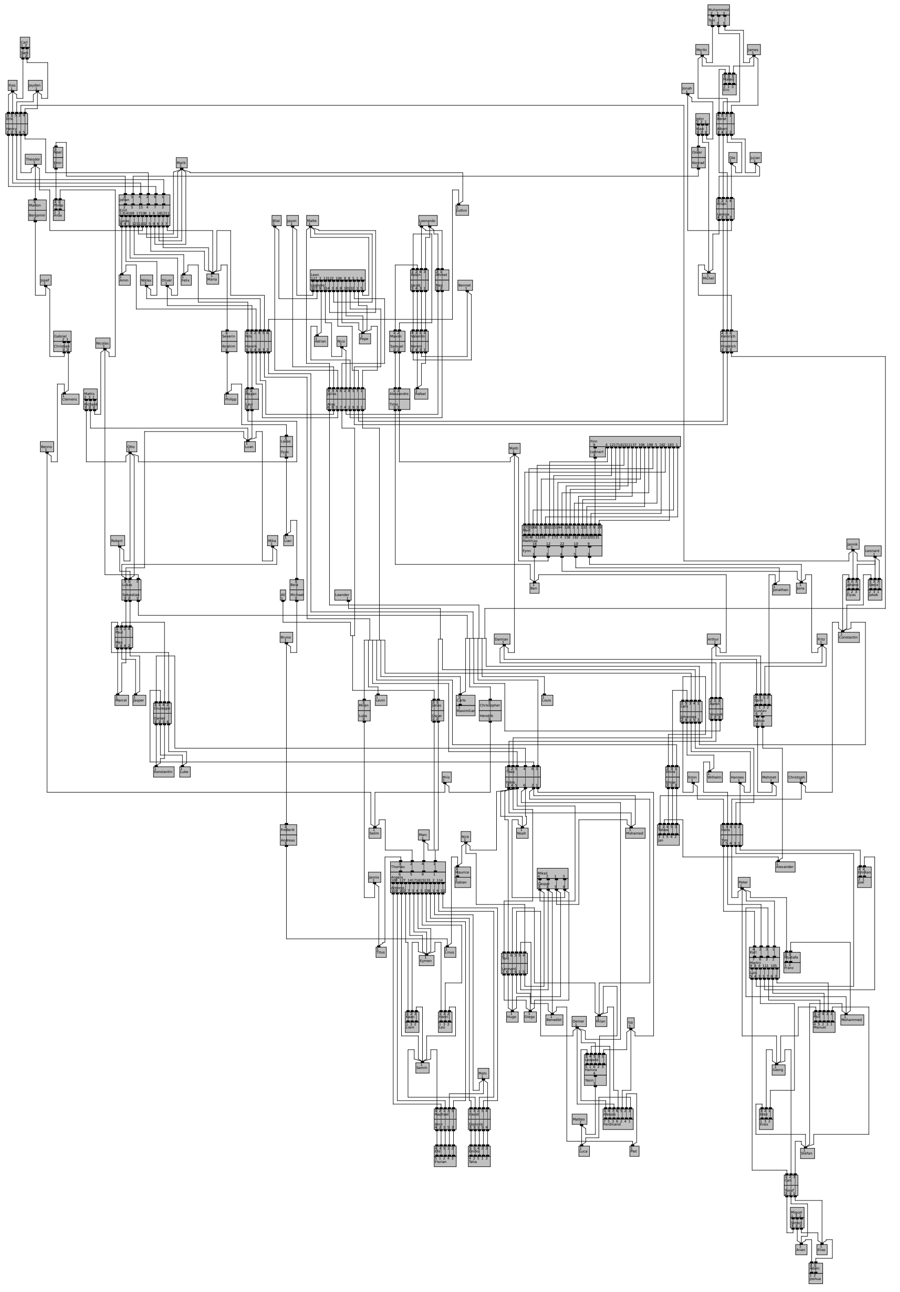


Figure 7.23: Original cable plan (anonymized) from the readable data set with 225 vertices and 562 crossings drawn using Kieler.

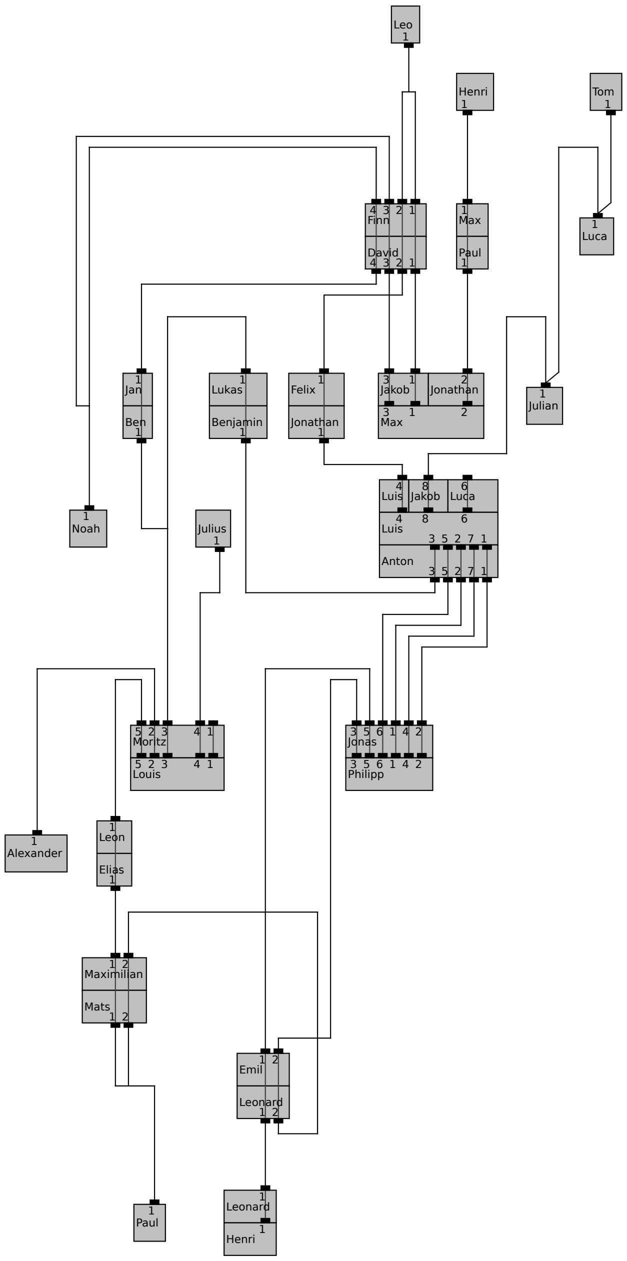


Figure 7.24: Pseudo cable plan generated from the readable data set with 39 vertices and three crossing drawn by our algorithm using FD, PORTS, and RELPOS.

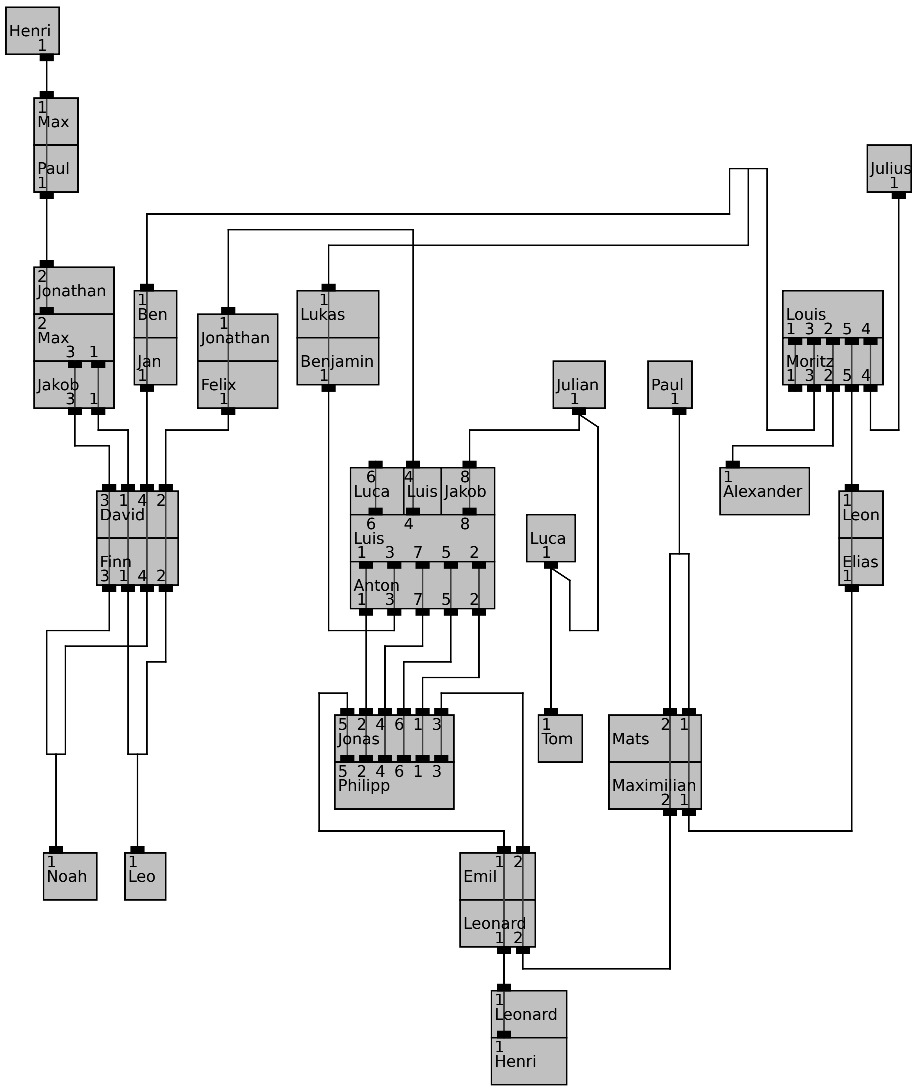


Figure 7.25: Pseudo cable plan generated from the readable data set with 39 vertices and three crossings drawn using KIELER.

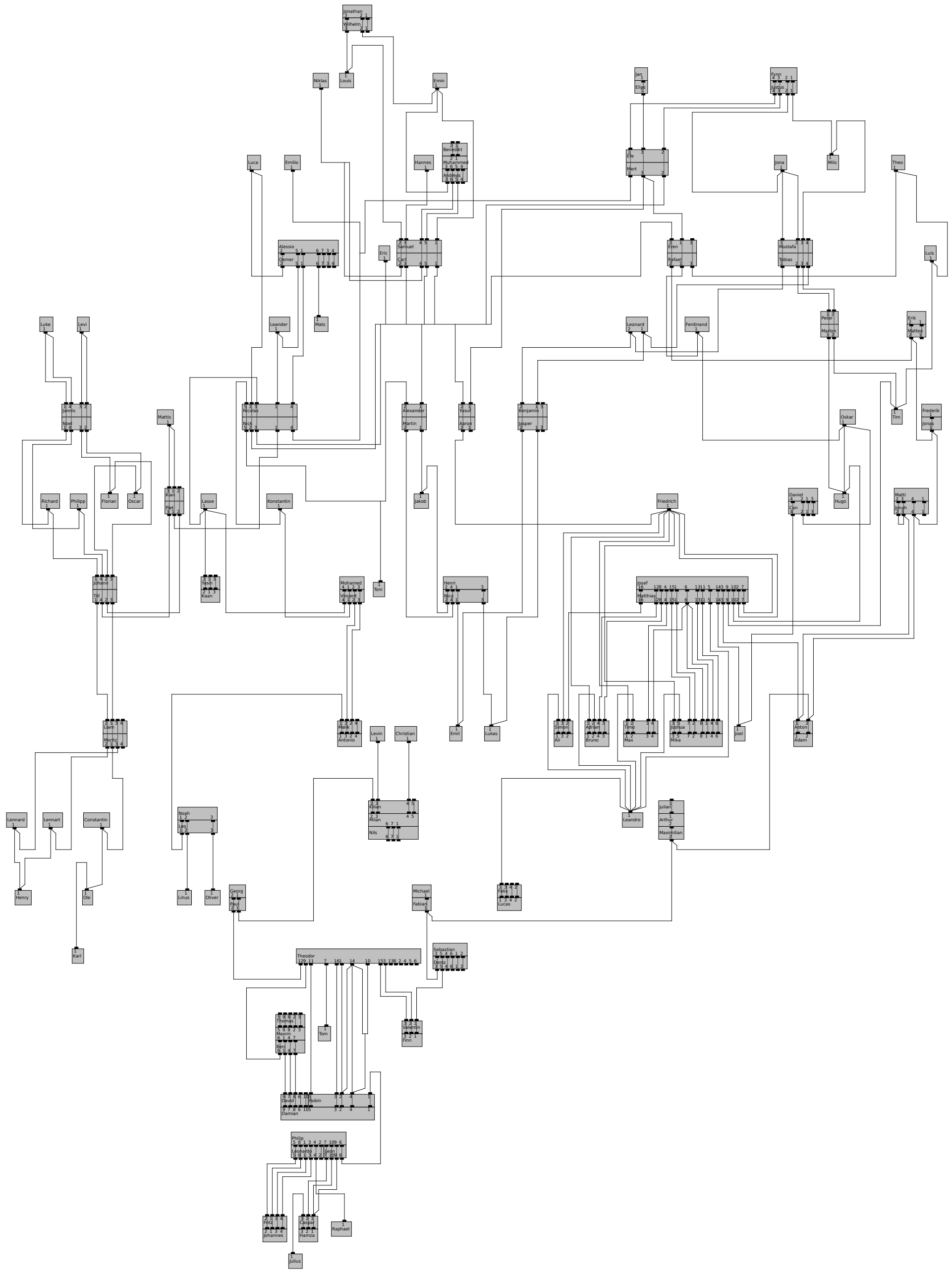


Figure 7.26: Pseudo cable plan generated from the readable data set with 144 vertices and 86 crossings drawn by our algorithm using FD, Ports, and RelPos.

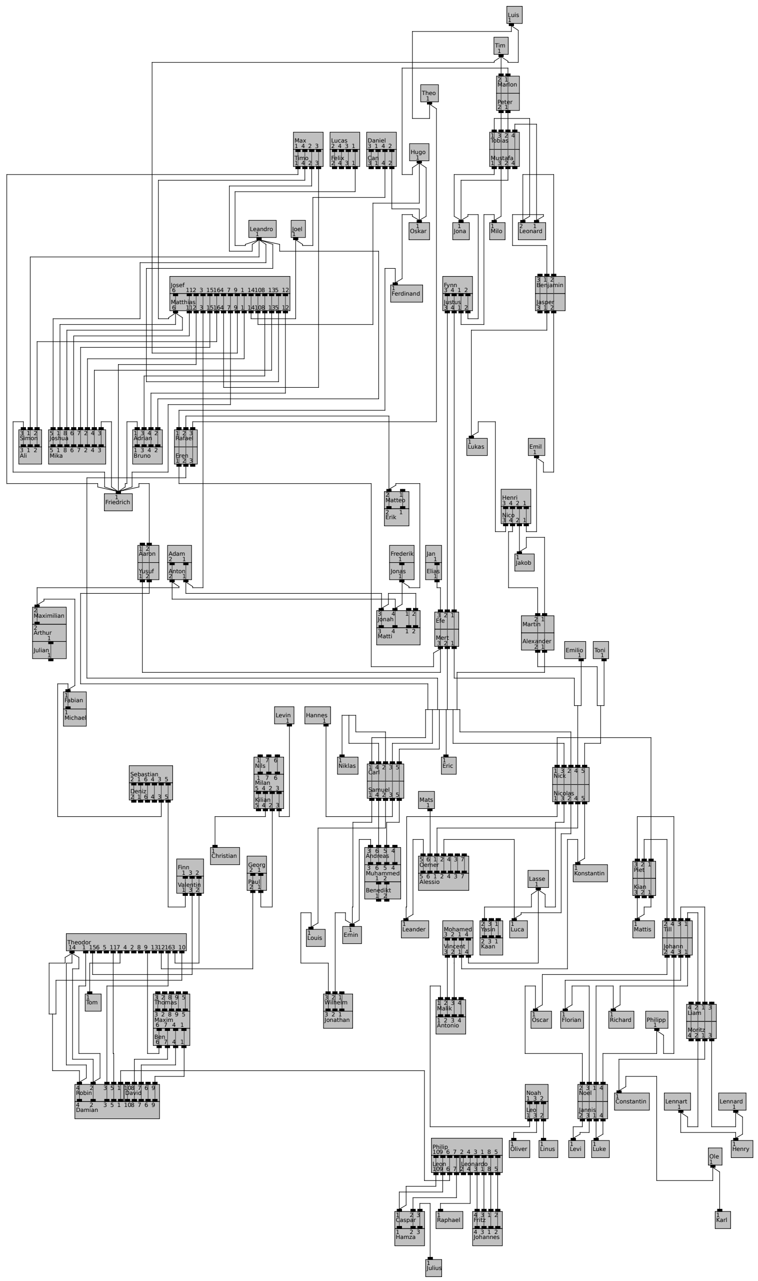


Figure 7.27: Pseudo cable plan generated from the readable data set with 144 vertices and 157 crossings drawn using Kieler.

Part III

Simplifying Polylines

Chapter 8
Faster Polyline Simplification under the Local Fréchet Distance

Visualization of geographical information is a task of high practical relevance. A recent example is the implementation of online maps. Such maps are most helpful if the information is neatly displayed and can be grasped quickly and unambiguously. This often means that the full data needs to be filtered and abstracted. Many important elements in maps like borders, streets, rivers, or trajectories are displayed as polylines. For such a polyline, a simplification is supposed to be as sparse as possible and as close to the original as necessary. This brings us to the topic of the last part of this book, which we start by improving the running time of a classic polyline simplification algorithm.

8.1 Introduction

Polyline simplification has a long history in computational geometry, where it has also been known as *polygonal approximation*, *line generalization*, or *ε-simplification*. It owes its relevance – also beyond computational geometry – to a large variety of applications, such as processing of vector graphics [WM03, vKLW20], robotics [DH73, NGM$^+$07], trajectory clustering [BBK$^+$20], shape analysis [MCW$^+$19], data compression [MdB04], curve fitting [Ram72], and map visualization [VW90, AW12, Ise13, AKPW15, GMA$^+$15].

The task of polyline simplification is to replace a given polyline on n vertices with a minimum-size subsequence of its vertices while ensuring that the input and the output polyline are sufficiently similar. The similarity is governed by a given distance threshold δ. Line segments between vertices in the output polyline are *shortcuts*. To determine the similarity of the input and output polyline, the Hausdorff and the Fréchet distance are the most commonly used measures. Both can be applied either globally or locally. In the global version, the distance between the entire input and output polyline is measured and must not exceed δ. In the local version, the distance between each shortcut and the part of the input polyline it bridges must not exceed δ.

Previous Work. For many applications, local similarity is more sensible and intuitive. Simplifications with global similarity have only been studied recently. For the global (undirected) Hausdorff distance, computing a simplified polyline with the smallest number of shortcuts yields an NP-hard problem [vKLW20]. For the global Fréchet distance, an $\mathcal{O}(n^3)$-time algorithm was designed by Bringmann and Chaudhury [BC21]. For the more extensively studied and more classical problem of simplification with the local Hausdorff distance, the Imai–Iri algorithm [II88], published in 1988, guarantees a running time of $\mathcal{O}(n^3)$ by reducing the simplification problem to a graph problem. Essentially, the algorithm constructs a graph where the polyline vertices are the nodes and where there is an edge between a pair of vertices if they can be connected with a shortcut respecting the δ-distance bound. For the local Hausdorff distance, Melkman and O'Rourke [MO88] showed already in 1988 that the Imai–Iri algorithm can be improved to run in $\mathcal{O}(n^2 \log n)$ time by making the graph construction phase more efficient. In 1996, Chan and Chin [CC96] further reduced the running time to $\mathcal{O}(n^2)$.

For the local Fréchet distance, though, the cubic running time of the Imai–Iri algorithm, which was shown by Godau [God91] in 1991, was a longstanding bound being referenced also in recent publications [vKLW20, BC21]. Agarwal, Har-Peled, Mustafa and Wang [AHMW05] explicitly posed the problem of whether there exists a subcubic algorithm for polyline simplification under the local Fréchet distance as an open question in 2005. The question was answered positively very recently by Buchin, van der Hoog, Ophelders, Schlipf, Silveira, and Staals [BvdHO+22]. They describe a data structure that outputs the Fréchet distance between any line segment and any subpolyline of a preprocessed input polyline in $\mathcal{O}(\sqrt{n}\log^2 n)$ time. Using this data structure to check whether there is a valid shortcut for every pair of vertices, a polyline simplified optimally can be computed in $\mathcal{O}(n^{5/2+\varepsilon})$ time (and space) for any $\varepsilon > 0$. We remark that this data structure is quite sophisticated and actually more powerful than required for polyline simplification. To check whether there is a valid shortcut between a pair of vertices with respect to the Fréchet distance, it suffices to be able to decide whether the distance of the shortcut to its subpolyline is at most δ. However, the data structure returns the exact distance value and it can accomplish this for arbitrary line segments and not only potential shortcuts.

Related Work. The most practically relevant setting for polyline simplification is to consider two-dimensional input curves in the Euclidean plane (i.e., under the L_2-norm). However, the problem was also studied in higher dimensions $d > 2$ and under different norms. Table 8.1 provides an overview of known lower and upper bounds for optimal polyline simplification.

The $\mathcal{O}(n^3)$-time algorithms by Imai, Iri, and Godau [II88, God91] for polyline simplification under the local Hausdorff and Fréchet distance as well as the $\mathcal{O}(n^3)$-time algorithm by Bringmann and Chaudhury [BC21] for the global Fréchet distance can be generalized to work in $\mathbb{R}^{d\geq 2}$ with the running time only increasing by a polynomial factor in d. For the local Hausdorff distance in two dimensions, Chan and Chin [CC96] showed that the Imai–Iri algorithm can be improved to run in $\mathcal{O}(n^2)$ time for the L_1-, L_2- and L_∞-norms (the concept can also be applied to any L_p-norm with $p \in (1, \infty)$, up to possible numerical issues that are further discussed in Section 8.4.3). In L_1, this has been improved to $\mathcal{O}(f(\varepsilon)n^{4/3+\varepsilon})$ for any $\varepsilon > 0$ by Agarwal and

norm	dim.	lower bound	local Hausdorff	local Fréchet	global F.
L_1	2		$\mathcal{O}(f(\varepsilon)n^{4/3+\varepsilon})$ [AV00] $\mathcal{O}(n^2)$ [CC96]	$\mathcal{O}(n^2)$ T8.15	$\mathcal{O}(n^3)$ [BC21]
	d	$\hat{\mathcal{O}}(n^{3-\varepsilon})$ [BC21]	$\hat{\mathcal{O}}(2^d n^2)$ [BCD+02] $\hat{\mathcal{O}}(n^3)$ [II88]	$\hat{\mathcal{O}}(n^3)$ [God91]	$\hat{\mathcal{O}}(n^3)$ [BC21]
L_p	2		$\mathcal{O}(n^2)$ [CC96]	$\mathcal{O}(n^2 \log n)$ C8.13	$\mathcal{O}(n^3)$ [BC21]
$p \in (1,\infty)$ $p \neq 2$	d	$\hat{\mathcal{O}}(n^{3-\varepsilon})$ [BC21]	$\hat{\mathcal{O}}(n^3)$ [II88]	$\hat{\mathcal{O}}(n^3)$ [God91]	$\hat{\mathcal{O}}(n^3)$ [BC21]
L_2	2		$\mathcal{O}(n^2)$ [CC96]	$\mathcal{O}(n^2 \log n)$ T8.12 $\mathcal{O}(n^{5/2+\varepsilon})$ [BvdHO+22]	$\mathcal{O}(n^3)$ [BC21]
	3		$\mathcal{O}(n^2 \log n)$ [BCD+02]	$\mathcal{O}(n^3)$ [God91]	$\mathcal{O}(n^3)$ [BC21]
	d	$\hat{\mathcal{O}}(n^{2-\varepsilon})$ [BBK+16]	$\hat{\mathcal{O}}(n^3)$ [II88]	$\hat{\mathcal{O}}(n^3)$ [God91]	$\hat{\mathcal{O}}(n^3)$ [BC21]
L_∞	2		$\mathcal{O}(n^2)$ [CC96]	$\mathcal{O}(n^2)$ T8.15	$\mathcal{O}(n^3)$ [BC21]
	d	$\hat{\mathcal{O}}(n^{2-\varepsilon})$ [BBK+16]	$\hat{\mathcal{O}}(n^2)$ [BCD+02]	$\hat{\mathcal{O}}(n^3)$ [God91]	$\hat{\mathcal{O}}(n^3)$ [BC21]

Table 8.1: Conditional lower bounds (given as running times that are excluded) and algorithmic upper bounds for polyline simplification in $\mathbb{R}^d$ under different similarity measures, L_p-norms, and dimensions. Here, n is the number of vertices, d the dimension and ε is any positive constant. The $\hat{\mathcal{O}}$-notation hides polynomial factors in d. Blue entries mark the results shown in this chapter.

Varadarajan [AV00] using a more compact representation of the shortcut graph.[16] Furthermore Barequet et al. [BCD+02] proposed an $\mathcal{O}(n^2 \log n)$-time algorithm for the local Hausdorff distance under the L_2-norm which works in $\mathbb{R}^3$, as well as an $\mathcal{O}(d2^d n^2)$-time algorithm for the L_1-norm and an $\mathcal{O}(d^2 n^2)$-time algorithm for the L_∞-norm.

Bringmann and Chaudhury [BC21] have also proven a conditional lower bound for simplification in $\mathbb{R}^d$ under the local Hausdorff distance as well as under the local and global Fréchet distance. More precisely, for every L_p-norm with $p \in [1, \infty), p \neq 2$, algorithms with a running time subcubic in n and polynomial in d were ruled out (unless the so-called $\forall\forall\exists$-OV hypothesis fails, which is not very likely). For large dimensions d, the algorithmic upper bounds of $\mathcal{O}(n^3 \cdot \text{poly}(d))$ for polyline simplification discussed above are hence tight. However, the lower bound still allows the existence of simplification algorithms with a running time in $\mathcal{O}(n^k \cdot \exp(d))$ with $k < 3$. Hence, for small values of d (which are of high practical relevance), faster algorithms are possible, as evidenced by the $\mathcal{O}(d2^d n^2)$-time algorithm for the local Hausdorff distance under the L_1-norm [BCD+02]. For the L_2- and L_∞-norms, the best currently known conditional lower bound for three of the similarity measures – local Hausdorff distance, local Fréchet distance, and global Fréchet distance – was proven by Buchin et al. [BBK+16]. Their proof rules out algorithms with a subquadratic running time in n and polynomial running time in d (unless the so-called strong exponential time hypothesis fails). Here again, better running times for simplification problems in a low-dimensional space with $d \in o(\log n)$ are still possible.

[16] The shortcut graph is defined in Section 8.2.1.

As the cubic running time of the Imai–Iri algorithm and the quadratic running time of the Chan–Chin algorithm may be prohibitive for processing long polylines even for $d = 2$, heuristics and approximation algorithms have been investigated. The Douglas–Peucker algorithm from 1973 [DP73] (also discovered by Ramer in 1972 [Ram72]) is one of the simplest and widely-used heuristics. It computes a simplified polyline under the local Hausdorff distance in $\mathcal{O}(n \log n)$ time [HS92] and under the local Fréchet distance in $\mathcal{O}(n^2)$ time [vKLW20] – but without any guarantee regarding the solution size.

There are other heuristics neither based on the Hausdorff nor the Fréchet distance like the algorithm by Visvalingam and Whyatt [VW93], which measures the importance of a vertex by the triangular area it adds.

Agarwal et al. [AHMW05] presented an approximation algorithm with a running time of $\mathcal{O}(n \log n)$ that works for any L_p-norm and generalizes to $\mathbb{R}^d$. It computes a simplification under the local Fréchet distance for δ, where the length of the simplified polyline does not exceed the length of the optimally simplified polyline for $\delta/2$.

There are also variants of the Fréchet distance that allow for faster polyline simplification. For example, polyline simplification under the discrete Fréchet distance (where only the distance between the vertices but not the points on the line segments in between matters) can be solved to optimality in $\mathcal{O}(n^2)$ time [BJW$^+$08]. However, the discrete Fréchet distance heavily depends on the density of vertices on the polyline, and hence for many applications the continuous Fréchet distance studied in this book constitutes a more meaningful measure.

The problem variant where the requirement is dropped that all vertices of the simplification must be vertices of the input polyline is called a *weak* simplification. Guibas et al. [GHMS93] showed that an optimal weak simplification under the (global) Fréchet distance with distance threshold δ can be computed in $\mathcal{O}(n^2 \log^2 n)$ time. Later Agarwal et al. [AHMW05] gave an $\mathcal{O}(n \log n)$-time approximation algorithm for a weak simplification violating the distance threshold δ by a factor of at most 8; see also an overview by Van de Kerkhof et al. [vdKKL$^+$19].

Regarding our techniques, we remark that the concept of a *wavefront* being comprised of (circular) arcs is well-established in computational geometry. Mitchell, Mount, and Papadimitriou [MMP87] have introduced the *continuous Dijkstra* method in 1987, where a wavefront is expanded[17] along a surface of a polyhedron to allow shortest path computations. This method was also used by Hershberger and Suri [HS99] to compute a shortest path in the plane given a set of polygonal obstacles. Also, many sweep-line algorithms maintain a kind of a wavefront. For example, the famous algorithm by Fortune [For87] for computing a Voronoi diagram maintains a wavefront made up of hyperbolic curves. Another approach computes a (weighted) Voronoi diagram directly by a wavefront expanding around the input points [HdL20].

Apart from a few similarities regarding the computation of intersection points of (circular) arcs, line segments, etc., we use a different type of wavefront. In all of the aforementioned examples, the wavefront is a *kinetic* data structure where things move or expand continuously (although it suffices to consider a few discrete events).

[17] This expansion is realized step-wise by discrete events.

In contrast, the structure by Melkman and O'Rourke [MO88] that we call a wavefront in this chapter is more static and simpler: it is a collection of (circular) arcs that we update iteratively with a new unit disk. There is no expansion of existing arcs over time.

Contribution. We present an algorithm for polyline simplification under the local Fréchet distance in two dimensions and for several L_p-norms with a (near-)quadratic running time; see Section 8.3.

Our algorithm heavily builds upon the Melkman–O'Rourke algorithm [MO88]. They exploit the geometric properties of the local Hausdorff distance using cone-shaped *wedges* and a *wavefront* to accelerate the shortcut graph construction. We adapt both of these concepts to the local Fréchet distance. We carefully study the properties of the resulting wavefront and explain how to maintain and efficiently update a wavefront data structure, which at its core is a simple balanced binary search tree. As our main result, we prove that the asymptotic running time of the Melkman–O'Rourke algorithm does not increase with our modifications, and hence optimal simplifications under the local Fréchet distance can be computed in $\mathcal{O}(n^2 \log n)$ time using $\mathcal{O}(n)$ space; see Section 8.4.

This is a large improvement compared to the cubic running time by Imai and Iri and by Godau but also with respect to the currently best running time of $\mathcal{O}(n^{5/2+\varepsilon})$ [BvdHO+22]. Compared to the algorithm by Buchin et al. [BvdHO+22] ours is much simpler. It is also faster than the $\mathcal{O}(n^2 \log^2 n)$ running time of the weak simplification algorithm by Guibas et al. [GHMS93, Theorem 14] by a logarithmic factor. However, we remark that parts of their algorithm [GHMS93, Def. 4, Theorem 7, Lemma 8, Lemma 9] can be used to tackle the problem under the local Fréchet distance, and we do partially re-use their techniques. Yet, their procedure is more complicated since it maintains more geometric information only needed for weak simplifications. Our algorithm is hence more straight-forward for the setting of polyline simplification under the local Fréchet distance, which makes it conceptually easier to understand.

Besides this new application for the local Fréchet distance, investigating the structure and implementation of *wedges* and *wavefronts* is of independent interest and may also help to understand better the work by Melkman and O'Rourke [MO88] from 1988 and the algorithm by Guibas et al. [GHMS93], who both employ these data structures but give little detail on its structural properties and on how to perform operations with the wavefront.

Consequently, we show that under the L_1- and L_∞-norms, the wavefront has constant complexity, which improves the running time to $\mathcal{O}(n^2)$. We argue that for a natural class of polylines, a quadratic running time can be achieved as well. To this end, we introduce the concept of *ν-light* polylines; see Section 8.5.

We start, however, with some preliminaries in Section 8.2. There, we provide more details on existing polyline simplification algorithms we build upon. Furthermore, we give some more definitions and we fix our notation.

8.2 Preliminaries

We first sketch the polyline simplification algorithms by Imai and Iri [II88], Melkman and O'Rourke [MO88], and Guibas et al. [GHMS93]. They provide the main ingredients for our algorithm. Afterwards we specify our notation and give some additional definitions that we need in this chapter.

8.2.1 The Imai–Iri Algorithm

Given an n-vertex polyline $L = \langle p_1, \dots, p_n \rangle$, the polyline simplification algorithm by Imai and Iri [II88] proceeds in two phases. In the first phase, the *shortcut graph* is constructed. This graph has a node for each vertex of L, and it has an edge between two nodes if and only if there is a valid shortcut between the corresponding two vertices of L. For the Hausdorff and the Fréchet distance, it can be checked in linear time whether the distance between a line segment and a polyline exceeds δ [AG95]. Hence, the total running time of the first phase amounts to $\mathcal{O}(n^3)$. In the second phase, a shortest path from the first node p_1 to the last node p_n is computed in the shortcut graph, which can be accomplished in $\mathcal{O}(n^2)$ time. In a naive implementation, the space consumption is in $\mathcal{O}(n^2)$. However, it is not necessary to first construct the full shortcut graph and to compute the shortest path subsequently. Instead, the space consumption can be reduced to $\mathcal{O}(n)$ by interleaving the two phases as follows: For $i \in \{1, \dots, n\}$, the shortest path distance d_i from p_i to p_n can be computed in linear time by considering the set of all valid shortcuts X from p_i to p_j with $j \in \{i+1, \dots, n\}$ and setting $d_i = 1 + \min_{\langle p_i, p_j \rangle \in X} d_j$. Hence, if the vertices are traversed in reverse order, only the distance values for already processed vertices and the shortcuts of the currently considered vertex need to be kept in memory to compute the correct solution without increasing the asymptotic running time.

8.2.2 The Melkman–O'Rourke Algorithm

Since in the Imai–Iri algorithm the construction of the shortcut graph dominates the runtime, accelerating this first phase also leads to an overall improvement. Melkman and O'Rourke [MO88] introduced a faster technique to compute the shortcut graph for the local Hausdorff distance. Starting once for each $i \in \{1, \dots, n\}$ at vertex p_i, they traverse the rest of the polyline vertex by vertex in $\mathcal{O}(n \log n)$ time to determine all valid shortcuts originating at p_i.

To this end, they maintain a cone-shaped region called *wedge* in which all valid shortcuts necessarily lie. When traversing the polyline, the wedge may become narrower iteratively. Moreover, they maintain a *wavefront*,[18] which is a sequence of circular arcs of unit disks. The wavefront subdivides the wedge into two regions – a valid shortcut $\langle p_i, p_j \rangle$ has the endpoint p_j in the region not containing p_i. In other

[18] Melkman and O'Rourke [MO88] use the term *frontier* instead of wavefront. Within the cone, they only call the region on the other side of the frontier *wedge* and they call the associated data structure *wedge data structure*. Our notation to call the whole cone *wedge* is in line with the algorithm by Chan and Chin [CC96].

words, a valid shortcut needs to cross the wavefront. The wavefront has size $\mathcal{O}(n)$ and is stored in a balanced search tree (in the original article an augmented 2-3-tree) such that querying and updating operations can be done in amortized $\mathcal{O}(\log n)$ time.

Containment in the wedge can be checked in constant time and the position of a vertex relative to the wavefront can be determined in $\mathcal{O}(\log n)$ time. Updating the wedge can be done in constant time. Updating the wavefront may involve adding an arc and removing several arcs. Here, the crucial observation [MO88] is that the order of arcs on the wavefront is reverse to the order of the corresponding unit disk centers – all with respect to the angle around p_i. This allows for binary search in $\mathcal{O}(\log n)$ time to locate a new arc within the wavefront. Although a linear number of arcs may be removed from the wavefront in a single step, over all steps any arc is removed at most once. Amortized, this results in a running time of $\mathcal{O}(n \log n)$ per starting vertex p_i and $\mathcal{O}(n^2 \log n)$ in total.

8.2.3 Algorithm by Guibas, Hershberger, Mitchell, and Snoeyink

Guibas et al. [GHMS93] study weak polyline simplification. There, given an n-vertex polyline $L = \langle p_1, \dots, p_n \rangle$ and a distance threshold δ, the objective is to compute any polyline $S = \langle q_1, \dots, q_m \rangle$ of smallest possible length m that hits all unit disks around the vertices in L in the given order, which they call *ordered stabbing.* To additionally have Fréchet distance at most δ between L and S, each vertex q_j of S needs to be in distance $\leq \delta$ to some point of c_L. They describe a 2-approximation algorithm running in $\mathcal{O}(n^2 \log n)$ time and a dynamic program solving this problem exactly in $\mathcal{O}(n^2 \log^2 n)$ time.

Both algorithms essentially rely on a subroutine to decide whether there exists a line ℓ (a *stabbing line*) that intersects a given set of n ordered unit disks $\langle D_1, \dots, D_n \rangle$ such that ℓ hits some points $\langle r_1, \dots, r_n \rangle$ with $r_i \in D_i$ for $i \in \{1, \dots, n\}$ in order [GHMS93, Def. 4]. This subroutine runs in $\mathcal{O}(n \log n)$ time [GHMS93, Lemma 9]. It is based on an algorithm computing iteratively two hulls and two limiting lines through the unit disks that describe all stabbing lines [GHMS93, Algorithm 1]. They also maintain the wavefront as described in the Melkman–O'Rourke algorithm. However, they add an update step to ensure that the stabbing line respects the order of the unit disks.[19] We use conceptually the same update step and explain it in more detail in Sections 8.3.1 and 8.4.2.

To compute an optimal weak simplification S with the dynamic program, this subroutine is called once per unit disk induced by vertices in L. Guibas et al. remark that the wavefront might have non-constant complexity. Hence they refrain from storing it explicitly. They only keep the wedge and support information in memory and construct the remaining information necessary to perform the update steps on demand. This further complicates the algorithm and adds a logarithmic factor per unit disk to the overall running time, which then is in $\mathcal{O}(n^2 \log^2 n)$.

[19] This is necessary because here the Fréchet distance is considered, while Melkman and O'Rourke only considered the Hausdorff distance.

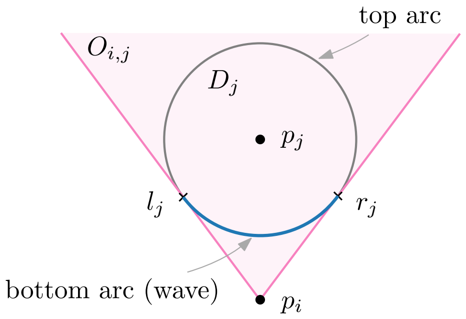


Figure 8.1: Unit disk and local wedge around a vertex p_j with respect to a vertex p_i.

8.2.4 Definitions and Notation

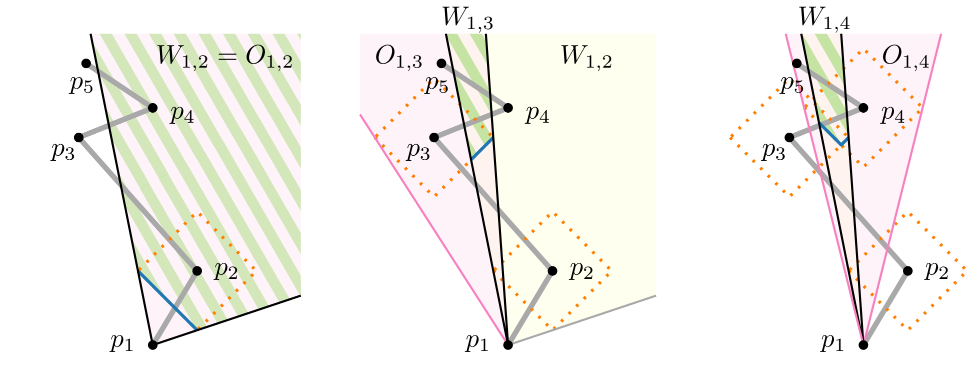
The following definitions are illustrated in Figures 8.1 and 8.2. Let D_j be the unit disk around p_j. When starting at p_i and encountering p_j during the traversal, we denote by $O_{i,j}$ the *local wedge* of p_i and p_j, that is, the area between the two tangential rays of D_j emanating at p_i. We let l_j (r_j) be the left (right)[20] tangential point of D_j and $O_{i,j}$. Between l_j and r_j, there are two arcs of the boundary of D_j – the *bottom arc* and the *top arc*[21] (of D_j). Clearly, any ray emanating at p_i intersects the bottom and the top arc at most once each. We call the bottom arc of the boundary of D_j between l_j and r_j the *wave* of $O_{i,j}$. We call the region within $O_{i,j}$ and above and on its wave the *local valid region* of $O_{i,j}$.

The (global) wedge $W_{i,j}$ is an angular region having its origin at p_i. We define $W_{i,i}$ to be the whole plane and each $W_{i,j}$ for $j > i$ is essentially the intersection of all local wedges up to $O_{i,j}$. We remark that, as mentioned in Section 8.2.3, we apply an extra update step specific to the Fréchet distance that we describe in Section 8.3.1. This update step may narrow the wedge when obtaining $W_{i,j}$ from $W_{i,j-1}$. Therefore, $W_{i,j} \subseteq \bigcap_{k \in \{i+1,i+2,\ldots,j\}} O_{i,k}$ holds. We give a precise inductive definition of the wedge $W_{i,j}$ when we describe the algorithm in Section 8.3.1.

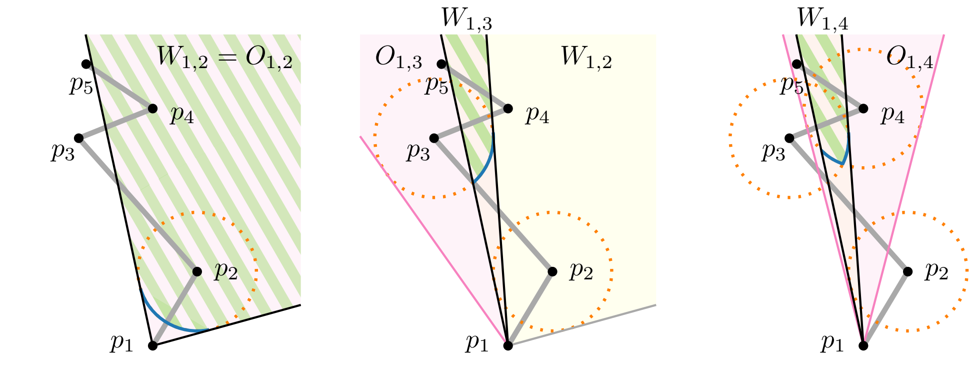
We call the region within $W_{i,j}$ and above and on the wavefront the *valid region* of $W_{i,j}$ (the valid region of $W_{i,i}$ is the whole plane). The wavefront itself is defined inductively. The *wavefront* of $W_{i,j}$ (for $j > i$) is the boundary of the intersection of the valid region of $W_{i,j-1}$ and the local valid region of $O_{i,j}$ within $W_{i,j}$ and excluding the boundary of $W_{i,j}$. Intuitively, it is the wavefront of $W_{i,j-1}$ within $W_{i,j}$ where we cut along the bottom arc of D_j; see Figure 8.2.

[20] W.l.o.g., we assume that p_i is below p_j and therefore at the bottom of $O_{i,j}$.

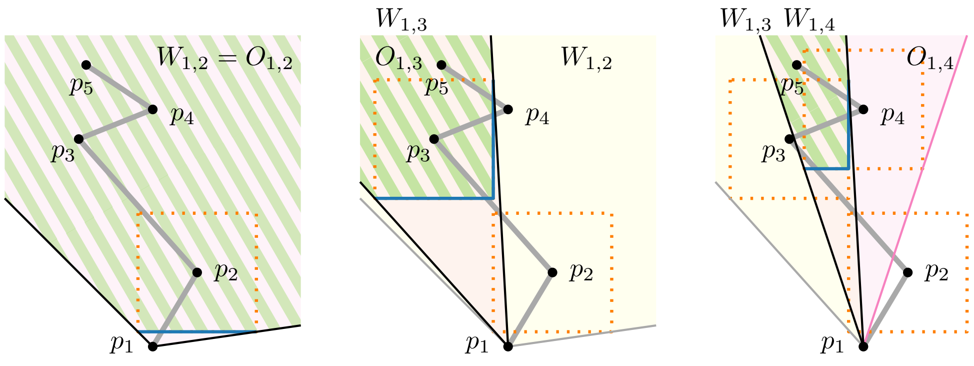
[21] W.l.o.g., we assume that p_{i+1} has distance at least δ to p_i because otherwise, we could ignore all vertices following p_i and having distance at most δ to p_i since they are in δ-distance to any shortcut $\langle p_i, p_j \rangle$. Note, though, that a vertex p_j with $j > i + 1$ could have distance at most δ to p_i. Then, we define $O_{i,j}$ as the whole plane and the whole boundary of D_j as its *top arc*.



(a) L_1-norm: unit disks are squares of side length $\sqrt{2}\delta$ rotated by 45 degrees w.r.t. the main axes.



(b) L_2-norm: unit disks are (circular) disks of radius δ. The wavefront comprises $\mathcal{O}(n)$ circular arcs.



(c) L_∞-norm: unit disks are squares of side length 2δ whose boundary is parallel to the main axes.

Figure 8.2: Iterative construction of the wedge and the wavefront: The intersections of the local wedges $O_{1,2}$, $O_{1,3}$, and $O_{1,4}$ (pink) determine here the wedges $W_{1,2}$, $W_{1,3}$, and $W_{1,4}$ (yellow). The wavefront (blue) is a sequence of unit disk arcs. Within the wedge and above the wavefront, there is the valid region (hatched green). If and only if a subsequent vertex p_j lies there, $\langle p_1, p_j \rangle$ is a valid shortcut. E.g., $\langle p_1, p_5 \rangle$ is a valid shortcut in the L_∞-norm, while in the L_1- and L_2-norms it is not.

8.3 Local-Fréchet Simplification Algorithm in Near-Quadratic Time

In this section, we describe how to obtain our algorithm for polyline simplification under the local Fréchet distance running in near-quadratic time by means of Melkman and O'Rourke [MO88], integrating ideas of Guibas et al. [GHMS93].

8.3.1 Outline

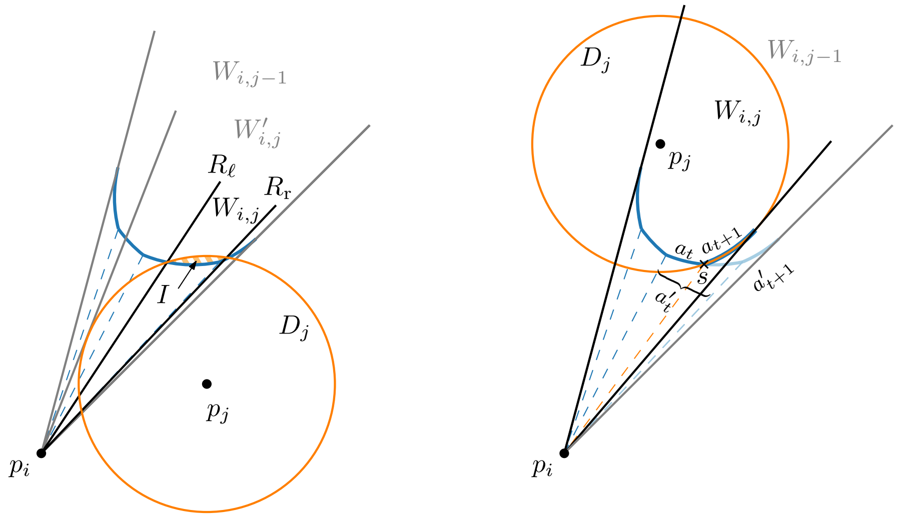
As all Imai–Iri based algorithms, we build the shortcut graph by traversing the given polyline n times – starting once from each vertex p_i for $i \in \{1, \ldots, n\}$ and determining all shortcuts starting at p_i. For each p_i, we construct a wedge with a wavefront, whose properties are analyzed in more detail in Section 8.4.

Next, we describe how to determine, for each vertex p_i, the set of subsequent vertices to which p_i has a valid shortcut. We traverse the polyline in order $p_{i+1}, p_{i+2}, \ldots, p_n$. During this traversal, we maintain the wedge in which all valid shortcuts need to lie. This would, as in the algorithm by Chan and Chin [CC96], suffice to assure that the Hausdorff distance threshold is not violated (which is a lower bound for the Fréchet distance). To also not exceed the Fréchet distance threshold, we use the wavefront. As in the algorithm by Melkman and O'Rourke, the invariant maintained is that for a valid shortcut from p_i to p_j with $j > i$, the vertex p_j has to be within the valid region of the wedge $W_{i,j-1}$. In this case, we add the directed edge $p_i p_j$ to the shortcut graph.

Then, regardless of whether $\langle p_i, p_j \rangle$ is a valid shortcut or not, we first update the wedge $W_{i,j-1}$ to an intermediate wedge $W'_{i,j}$ by computing the intersection between $W_{i,j-1}$ and the local wedge $O_{i,j}$. Afterwards, we update the intermediate wedge $W'_{i,j}$ to the wedge $W_{i,j}$ and we update the wavefront.

This update step is illustrated for the L_2-norm in Figure 8.3 and for multiple steps and multiple norms in Figure 8.2. For the L_2-norm and for the L_1- and L_∞-norms, we give more detail on this update step in Sections 8.4.2 and 8.5.1, respectively. It works as follows. A valid shortcut $\langle p_i, p_k \rangle$ with $k > j$ in the Fréchet distance needs to go through the intersection region I between the current valid region and the unit disk D_j around p_j. Otherwise, the vertices of the subpolyline from p_i to p_k would be encountered in the wrong order contradicting the definition of the Fréchet distance. Hence, we narrow the intermediate wedge $W'_{i,j}$ such that the rays R_ℓ and R_r emanating at p_i and enclosing I constitute the wedge $W_{i,j}$; see Figure 8.3a. This extra narrowing step is also applied by Guibas et al. in their line stabbing algorithm, but not by Melkman and O'Rourke. For the Hausdorff distance, it is irrelevant in which order the intermediate points of a shortcut are encountered by the shortcut segment.

Thereafter, we update the wavefront as Melkman and O'Rourke do. The part of the bottom arc of the unit disk D_j around p_j that is above the current wavefront is included into the new wavefront. Pictorially, the wavefront is moving upwards. For an example see Figure 8.3b. There, we compute the intersection point s between D_j



(a) When encountering p_j, we update the wedge in two steps – even if p_j lies outside the wedge.

(b) Vertex p_j contributes an arc to the new wavefront. Here, $\langle p_i, p_j \rangle$ is also a valid shortcut.

Figure 8.3: Updating the wedge and its wavefront in the L_2-norm.

and the wavefront and replace the arcs a'_t and a'_{t+1} of the wavefront by the arcs a_t (which is a part of a'_t) and a_{t+1} (which is a part of D_j). There can be up to two intersection points between D_j and the wavefront.

If during this process the valid region becomes empty, we abort the search for further shortcuts from p_i.

8.3.2 Correctness

To show that the algorithm works correctly, we prove two things: that all shortcuts the algorithm finds are valid (Lemma 8.5) and that the algorithm finds all valid shortcuts (Lemma 8.6). As it is more difficult to show this directly, we first state four helpful lemmas.

Lemma 8.1. *Let D and D' be two unit disks, and let p be a point that lies outside of D and D'. If the two bottom arcs (with respect to p) intersect, then the second intersection point is between their top arcs.*

Proof. For an illustration of this proof, see Figure 8.4. Without loss of generality, let p be below D and D', and let the center of D be to the left of the center of D'. Now the cone between the right tangent from p on D and the left tangent from p on D' contains all of the intersection area of D and D', and hence also both intersection points. We call the tangential points r_D and $l_{D'}$, respectively. Note that the case $r_D = l_{D'}$ is excluded as then D and D' would only have a single intersection point. For the intersection point s between the bottom arcs of D and D', we know that the

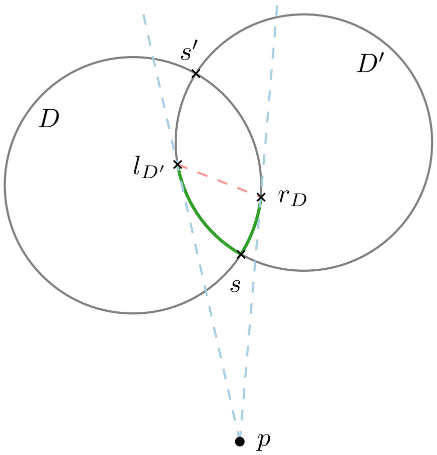


Figure 8.4: Illustration of the situation described in the proof of Lemma 8.1.

line segment $\overline{ps}$ does not intersect the inner part of any of the two disks by definition of the bottom arc. Hence the ray elongating this line segment has to go through the intersection area of D and D' above s. Therefore, the partial bottom arc of D from s to r_D and the partial bottom arc of D' from s to $l_{D'}$ are both on the boundary of the intersection area. As the intersection area is convex, it means that the line segment $\overline{l_{D'}r_D}$ is fully contained in the intersection area, and the intersection points have to be on opposite sites of the line through $l_{D'}$ and r_D. Accordingly, the second intersection point s' of D and D' then has to lie above $\overline{l_{D'}r_D}$ and is therefore on the respective top arcs of D and D'. □

Lemma 8.2. *Let $1 \leq i < j \leq k \leq n$. Consider the wavefront of $W_{i,k}$. If the unit disk D_j contributes an arc to the wavefront of $W_{i,k}$, then the wavefront of $W_{i,k}$ lies completely inside D_j.*

Proof. We argue that, for all $j \in \{i+1, \ldots, k\}$, the claim is true by considering first all arcs that had been added before and then all arcs that have been added after the arc of D_j was added to the wavefront.

Every arc $a_{j'}$ on the wavefront belonging to a vertex $p_{j'}$ with $j' < j$ lies inside D_j because when the wavefront of $W_{i,j}$ has been constructed, the wavefront of $W_{i,j}$ consisted of arcs of the wave of $O_{i,j}$, i.e., arcs of D_j, and it consisted of arcs of the wavefront of $W_{i,j-1}$ lying inside I, i.e., in the intersection between D_j and the valid region of $W_{i,j-1}$.[22]

Every arc $a_{k'}$ on the wavefront belonging to a vertex $p_{k'}$ with $j < k' \leq k$ lies completely inside D_j because if it were not, there would be an arc $a_{k'}$ (which is part of the bottom arc of the unit disk $D_{k'}$) that intersects D_j at s_1; see Figure 8.5. The

[22] We remark that even without the extra narrowing step using I, if D_j contributes an arc to the wavefront of $W_{i,j}$, then D_j contains the whole wavefront of $W_{i,j}$.

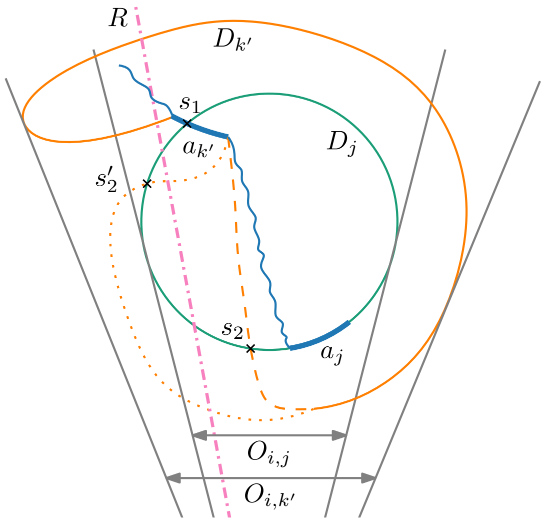


Figure 8.5: Configuration used to prove Lemma 8.2. The blue part is the wavefront including the arcs $a_{k'}$ and a_j. The disks D_j and $D_{k'}$ are unit disks and the gray rays indicate the local wedges.

intersection at s_1 is with the top arc of D_j as otherwise $a_{k'}$ would be (partially) outside the local valid region of $O_{i,j}$. For a_j to be in the local valid region of $O_{i,k'}$, D_j and $D_{k'}$ must intersect a second time. We consider two possible cases for a second intersection and denote them by s_2 and s_2'. First, assume that the intersection s_2 is between the bottom arc of $D_{k'}$ and the bottom arc of D_j. This however contradicts Lemma 8.1 because in s_1, the bottom arc of $D_{k'}$ was already involved. Hence, the second intersection point is s_2', which is an intersection between the bottom arc of $D_{k'}$ and the top arc of D_j. Then, however, there is a ray R originating in p_i that lies between s_1 and s_2' and intersects the bottom arc of $D_{k'}$ at least twice – a contradiction. □

Lemma 8.2 directly implies the following lemma.

Lemma 8.3. *Let q be a point lying on the wavefront of the wedge $W_{i,j}$. Then, $d(p_j, q) \leq \delta$.*

Lemma 8.4. *Let R be a ray emanating at p_i and lying inside the wedges $W_{i,j}$ and $W_{i,k}$ for some $i < j < k$. Moreover, let q_j and q_k be the intersection points between R and the wavefronts of $W_{i,j}$ and $W_{i,k}$, respectively. Then, $d(p_i, q_j) \leq d(p_i, q_k)$.*

Proof. Assume for contradiction that $d(p_i, q_j) > d(p_i, q_k)$. Then, q_k is below the wavefront of $W_{i,j}$ and, hence, q_k does not lie in the valid region of $W_{i,j}$ but in the valid region of $W_{i,k}$. However, the valid region of $W_{i,k}$ is the intersection of the local valid region of $O_{i,k}$ and all previous valid regions including $W_{i,j}$ and, thus, the valid region of $W_{i,k}$ is a subset of $W_{i,j}$. □

Putting Lemma 8.4 in other words, the wavefront may only move away but never towards p_i during the execution of the algorithm. We are now ready to prove the correctness of the algorithm by the following two lemmas.

Lemma 8.5. *Any shortcut found by the algorithm is valid under the local Fréchet distance and any L_p-norm with $p \in [1, \infty]$.*

Proof. Let $\langle p_i, p_k \rangle$ be a shortcut found by the algorithm. We show that there is a mapping of the vertices $\langle p_{i+1}, p_{i+2}, \ldots, p_{k-1} \rangle$ onto points $\langle m_{i+1}, m_{i+2}, \ldots, m_{k-1} \rangle$, such that $m_j \in \overline{p_i p_k}$ and $d(p_j, m_j) \leq \delta$ for every $j \in \{i+1, \ldots, k-1\}$, and m_j precedes or equals m_{j+1} for every $j \in \{i+1, \ldots, k-2\}$ when traversing $\overline{p_i p_k}$ from p_i to p_k. Clearly, this implies that also the Fréchet distance between each pair of line segments $\overline{p_j p_{j+1}}$ and $\overline{m_j m_{j+1}}$ is at most δ and, hence, $\langle p_i, p_k \rangle$ is a valid shortcut. In the remainder of this proof, we describe how to obtain $m_{i+1}, m_{i+2}, \ldots, m_{k-1} \in \overline{p_i p_k}$. To this end, we consider the wedge $W_{i,j}$ and the corresponding wavefront for each $j \in \{i+1, \ldots, k-1\}$, i.e., for each intermediate step when executing the algorithm. By construction of the algorithm, $\overline{p_i p_k}$ lies inside the wedge $W_{i,j}$ and p_k lies above its wavefront (since p_k lies in the valid region of $W_{i,k-1}$ and, by Lemma 8.4, the wavefront has never moved towards p_i). Let m_j be the intersection point of $\overline{p_i p_k}$ and the wavefront of $W_{i,j}$. By Lemma 8.3, $d(p_j, m_j) \leq \delta$. Moreover, by Lemma 8.4, m_j precedes or equals m_{j+1} for any $j \in \{i+1, \ldots, k-2\}$ when traversing $\overline{p_i p_k}$ from p_i to p_k. □

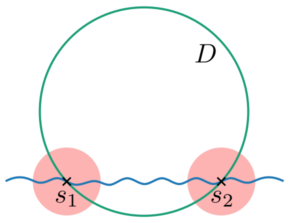
Lemma 8.6. *All valid shortcuts under the local Fréchet distance and any L_p-norm with $p \in [1, \infty]$ are found by the algorithm.*

Proof. Suppose for the sake of a contradiction that there is a valid shortcut $\langle p_i, p_k \rangle$ that was not found by the algorithm.

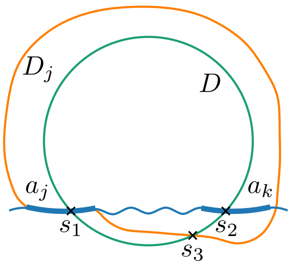
If p_k lay outside of $\bigcap_{j \in \{i+1, i+2, \ldots, k-1\}} O_{i,j}$, then there would be some $p_{j'}$ with $i < j' < k$ such that $d(p_{j'}, \overline{p_i p_k}) > \delta$. So, as in the algorithm by Chan and Chin [CC96], already the Hausdorff distance requirement would be violated and $\langle p_i, p_k \rangle$ would be no valid shortcut. Hence, p_k lies inside $\bigcap_{j \in \{i+1, i+2, \ldots, k-1\}} O_{i,j}$.

Suppose now that p_k lies inside $\bigcap_{j \in \{i+1, i+2, \ldots, k-1\}} O_{i,j}$ but outside $W_{i,k-1}$. W.l.o.g. p_k lies to the left of the wedge $W_{i,k-1}$. We know that there is some p_j with $i < j < k$ for which the extra narrowing step from Section 8.3.1 has been applied such that p_k lies to the left of $W_{i,j}$. For constructing $W_{i,j}$, we have considered the intersection area I between D_j and the wavefront of $W_{i,j-1}$. The left endpoint of I lies on the boundary of $W_{i,j}$ and is the intersection point between D_j and an arc of the wavefront of $W_{i,j-1}$ belonging to a vertex $p_{j'}$ with $i < j' < j$. Now consider the ray R that we obtain by extending $\overline{p_i p_k}$ at p_k. When traversing R, we first enter and leave the interior of D_j before we enter the interior of $D_{j'}$. Hence, the Fréchet distance between $\overline{p_i p_k}$ and $L[p_i, p_k]$ is greater than δ due to the order of $p_{j'}$ and p_j within $L[p_i, p_k]$. Therefore, p_k lies inside $W_{i,k-1}$.

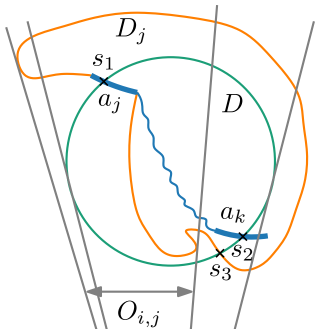
Finally, suppose that p_k lies inside $W_{i,k-1}$ but not in the valid region, i.e., p_k lies below the wavefront of $W_{i,k-1}$. Since p_k is below the wavefront, the line segment $\overline{p_i p_k}$ does not intersect the wavefront (otherwise, we would violate Lemma 8.8; see below).



(a) Situation that cannot occur.



(b) D intersects the wavefront twice with its bottom arc.



(c) D intersects the wavefront with its top and bottom arc.

Figure 8.6: Sketch for Lemma 8.7: a unit disk D intersects the wavefront (blue wavy line) twice.

Again, consider the ray R that we obtain by extending $\overline{p_i p_k}$ at p_k. Let the intersection point of R and the wavefront of $W_{i,k-1}$ be w. The point w lies on an arc of the wavefront. This arc is part of the bottom arc of a unit disk D_j belonging to some p_j with $i < j < k$. Since it is the bottom arc, p_k lies outside D_j and $d(p_k, p_j) > \delta$.

Therefore, p_k lies in the valid region of $W_{i,k-1}$. However, these are precisely the vertices for which the algorithm adds a shortcut. □

8.4 The Wavefront Data Structure

At the heart of the algorithm lies the maintenance of the wavefront. To show that the algorithm can be implemented to run in $\mathcal{O}(n^2 \log n)$ time, we next analyze the properties of the wavefront and discuss how to store and update it using a suitable (simple) data structure.

We start this section with a structural lemma, which seems rather special at first glance, but we employ it several times.

Lemma 8.7. *If a unit disk D intersects the wavefront more than once, then on the left side of the leftmost intersection point s_1 (relative to rays originating in p_i) and on the right side of the rightmost intersection point s_2, D is below the wavefront. In other words, the intersection pattern depicted in Figure 8.6a cannot occur.*

Proof. Clearly, if at s_1 the top arc of D intersects the wavefront, then on the left side of s_1, D is below the wavefront. Symmetrically, the same holds for s_2.

Now assume that at s_1 and at s_2, the bottom arc of D intersects the arcs a_j and a_k of the wavefront, respectively. We denote their unit disks by D_j and D_k. W.l.o.g. let D on the left side of s_1 be above the wavefront. By Lemma 8.2, D_j contains the rest of the wavefront including all of a_k. This means, that D intersects D_j at s_3 between s_1 and s_2 (potentially $s_2 = s_3$ if $D_j = D_k$); see Figure 8.6b. Because the intersection of D at s_2 is with the bottom arc of D, the intersection of D and D_j at s_3 is also with the bottom arc of D. This contradicts Lemma 8.1.

Finally, assume w.l.o.g. that at s_1 the top arc of D intersects the arc a_j of the wavefront and at s_2 the bottom arc of D intersects the arc a_k of the wavefront; see Figure 8.6c. Again by Lemma 8.2, the unit disk D_j of a_j contains the wavefront including the whole arc a_k. Hence, there is an intersection point s_3 of D and D_j between s_1 and s_2 (where $s_2 \neq s_3$ and $D_j \neq D_k$ as otherwise D and D_j would have an intersection between their bottom arcs and between a bottom and a top arc). At s_3 there is the bottom arc of D (since later at s_2, there is also the bottom arc of D involved). If D_j also would have its bottom arc at s_3, it would contradict Lemma 8.1. Therefore, at s_3, there is the top arc of D_j. This however means that s_3 is outside $O_{i,j}$ – a contradiction. □

8.4.1 Size of the Wavefront

We prove that the wavefront always has a size in $\mathcal{O}(n)$. This insight is based on the properties proven in the following lemmas.

Lemma 8.8. *Any ray emanating at p_i intersects the wavefront at most once.*

Proof. We prove this statement inductively. As $W_{i,i+1} = O_{i,i+1}$, consider the wave of $O_{i,i+1}$. Since the unit disk in any L_p-norm for $p \in [1, \infty]$ is convex, any ray emanating at p_i intersects a unit disk at most twice. The first intersection is with the bottom arc of the unit disk D_{i+1} and the second intersection is with the top arc of D_{i+1}. As the wave of $O_{i,i+1}$ is defined as the bottom arc of D_{i+1}, any ray emanating at p_i intersects the wave of $O_{i,i+1}$ at most once.

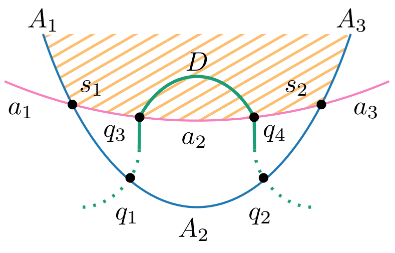
It remains to show the induction step for all $j > i+1$. By the induction hypothesis, we know that any ray emanating at p_i intersects the wavefront of $W_{i,j-1}$ at most once. The wavefront of $W_{i,j}$ is the boundary of the intersection of the valid region of $W_{i,j-1}$ and the local valid region of $O_{i,j}$. Consider a ray R originating at p_i. The ray R enters the valid region of $W_{i,j-1}$ at most at one point q where it also intersects the wavefront of $W_{i,j-1}$, and it enters the local valid region of $O_{i,j}$ at most at one point q' where it also intersects the wave of $O_{i,j}$. Hence, R enters the intersection of the valid region of $W_{i,j-1}$ and the local valid region of $O_{i,j}$ at most at one point – namely either at q or at q' (or $q = q'$). This is the only point of the wavefront of $W_{i,j}$ that is shared with R. □

We can make a similar statement for unit disks. The number of intersection points between a unit disk and the wavefront is important for updating the wavefront.

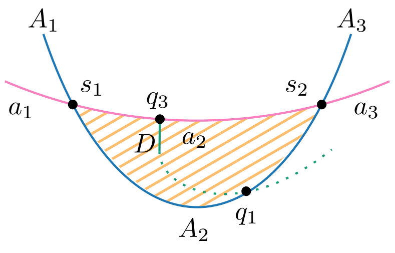
Lemma 8.9. *Any unit disk of radius δ intersects the wavefront at most twice.*

Proof. We prove this statement inductively. Say p_i is our start vertex and we consider the wavefront of $W_{i,i+1}$, which is the same as the wave of $O_{i,i+1}$, which is part of the boundary of a unit disk. Since each pair of unit disks in the L_p-norm for $p \in [1, \infty]$ intersects at most twice, we know that any unit disk intersects the wavefront of $W_{i,i+1}$ at most twice.

It remains to show the induction step for all $j > i+1$. Assume for a contradiction that a unit disk D intersects the wavefront of $W_{i,j}$ more than twice. Observe that



(a) Case A.



(b) Case B.

Figure 8.7: Cases in the proof of Lemma 8.9.

the wavefront of $W_{i,j}$ is a subset of the wavefront of $W_{i,j-1}$ and the wave of $O_{i,j}$. Say D intersects the wavefront of $W_{i,j-1}$ at q_1 and q_2 and D intersects the wave of $O_{i,j}$ at q_3 and q_4 (maybe one of these points does not exist.) Next, we argue topologically that at most two points of $\{q_1, q_2, q_3, q_4\}$ lie on the wavefront of $W_{i,j}$, which is a contradiction.

By the induction hypothesis, the wavefront of $W_{i,j-1}$ and the wave of $O_{i,j}$ intersect at most twice. Let these intersection points from left to right be s_1 and s_2; see Figure 8.7. Let the subdivisions of the wavefront of $W_{i,j-1}$ and the wave of $O_{i,j}$ induced by s_1 and s_2 be A_1, A_2, A_3 and a_1, a_2, a_3, respectively. Some of them may be empty. Clearly, the wavefront of $W_{i,j}$ is either A_1–a_2–A_3 or a_1–A_2–a_3. By Lemma 8.7, we know that it cannot be a_1–A_2–a_3, therefore, it is A_1–a_2–A_3.

Next, we analyze the intersection points q_3 and q_4 (maybe q_4 does not exist). Either one or two of them lies on a_2 as otherwise there are no more than two intersection points of D with the new wavefront.

Case A: The intersection points q_3 and q_4 lie on a_2; see Figure 8.7a. As both intersection points are between the unit disk D and the wave of $O_{i,j}$, i.e., a bottom arc of another unit disk, we know by Lemma 8.1 that q_3 and q_4 are contained in the top arc of D. Thus, there is no ray R to the left of q_3 or to the right of q_4 originating at p_i and intersecting the arc of D between q_3 and q_4 as otherwise R would intersect the top arc of D twice. Therefore, the arc of D between q_3 and q_4 lies in the valid region (hatched orange in Figure 8.7a) without reaching A_1 or A_3. When D passes through q_3 and q_4, it reaches the region between a_2 and A_2. If there are intersections between $W_{i,j-1}$ and D, they both lie on A_2.

Case B: Only one intersection point, let it be q_3, lies on a_2; see Figure 8.7b. If it is a touching point, then D lies in the region between a_2 and A_2 before and after reaching q_3 (because we can assume that both unit disks are non-identical). If it is an intersection point, then D passes through q_3 into the region between a_2 and A_2 (hatched orange in Figure 8.7b). To leave this region, q_1 (or q_2) lies on A_2. Hence, there are at most two points of $\{q_1, q_2, q_3, q_4\}$ on the new wavefront. □

From Lemma 8.9 it follows that in each step, the size of the wavefront increases at most by 2. This leads us to the following lemma.

Lemma 8.10. *The wavefront consists of at most $\mathcal{O}(n)$ arcs under any $L_{p\in(1,\infty)}$ norm.*

Proof. According to the inductive definition, we start with a wavefront consisting of one arc. Now in each step where we extend the wavefront, we consider the intersection between the current valid region and a local valid region – one is defined by the current wavefront, the other is defined by a single arc a. This is the intersection between the current wavefront and the unit disk on which a lies. By Lemma 8.9, we know that there are at most two intersection points. This means, the number of arcs on the wavefront increases by at most two. In the worst case, we start at vertex p_1 and adjust the wavefront $n-1$ times until we have created the wavefront of $W_{1,n}$. Therefore, any wavefront consists of at most $2n-3 \in \mathcal{O}(n)$ arcs.[23] □

8.4.2 Wavefront Maintenance under the L$_2$-Norm

In this section, we consider polyline simplification under the L$_2$-norm, i.e., the Euclidean norm. As there might be a linear number of arcs on the wavefront, we cannot simply iterate over all arcs in each step of the algorithm since this would require cubic time in total. Therefore, we employ a data structure that allows for querying, inserting, and removing an object in logarithmic time. Similar to Melkman and O'Rourke, we use a balanced search tree (e.g., a red-black tree) where we store the circular arcs [24] of the wavefront. The keys according to which the circular arcs are arranged in the search tree are the angles of their starting points with respect to p_i. These angles cover a range of less than π, hence, we may rotate the drawing when computing the wavefronts of a vertex p_i to avoid "jumps" from 2π to 0. We can then locate a point p_j relative to the wavefront and add or remove an arc on the wavefront in logarithmic time.

Note that, different from Melkman and O'Rourke and similar to Guibas et al., we have an additional update step where we determine the intersection region I and potentially make the wedge narrower. We show that we can update the wavefront in amortized logarithmic time using a simple case distinction. We compute the intersection area I only implicitly. For an overview, see Figure 8.8.

Say we are computing all valid shortcuts starting at p_i and we are currently processing a vertex p_j, which is the center of the unit disk D_j. We have already constructed the intermediate wedge $W'_{i,j}$ and clipped the wavefront of $W_{i,j-1}$ along the left and the right bounding rays R_ℓ and R_r of $W'_{i,j}$. For this clipping, we may have removed a linear number of arcs, however, over all iterations we remove every arc at most once. Now, both R_ℓ and R_r intersect D_j twice or touch D_j. Let q_1 and

[23] One can further observe that, by Lemma 8.7, the number of arcs on the wavefront increases actually by at most one per vertex p_j ($j \in \{2,\dots,n\}$). This means any wavefront consists of at most $n-1$ arcs.

[24] To represent a circular arc, we store its corresponding unit disk center and the points where the arc starts and ends.

q_2 denote the intersection points between R_ℓ and D_j (where q_1 is on the bottom arc of D_j). Similarly, let q_3 and q_4 denote the intersection points between R_r and D_j (where q_3 is on the bottom arc of D_j). Moreover, let l and r denote the intersection point between the wavefront and R_ℓ and between the wavefront and R_r, respectively.

The relative positions of l, q_1, and q_2 on R_ℓ and the relative positions of r, q_3, and q_4 on R_r determine where the intersection points s_1 and s_2 (if they exist) between D_j and the wavefront of $W_{i,j-1}$ can lie. (Recall that there are at most two intersection points by Lemma 8.9.) In the following, we write $a \prec b$ if a is below b along the ray R_ℓ or R_r. If $q_1 = l$ or $q_2 = l$, then we proceed as if R_ℓ was moved to the right by a tiny bit (symmetrically as if R_r was moved to the left). Thus, at such a point, the angle of the incident arc of the wavefront and D_j matters for the order. For the degenerate case $q_1 = l = q_2$, which includes a touching point between R_ℓ and D_j, we hence assume $q_1 \prec l \prec q_2$.

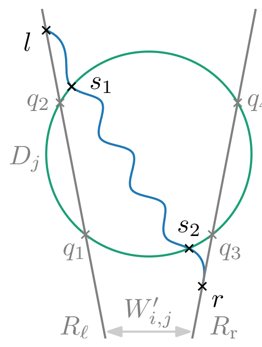
Next, we consider all orderings of l, r, q_1, q_2, q_3, and q_4. This gives rise to the following nine cases. We remark that two of them (Case **TB** and Case **BT**) cannot occur and two pairs of the remaining cases are symmetric, which leaves essentially five different cases.

(Case TB:) $q_1 \prec q_2 \prec l$ and $r \prec q_3 \prec q_4$; see Figure 8.8a. This case cannot occur. Suppose for a contradiction that we have this configuration. Then, there are precisely two intersection points s_1 and s_2 between the wavefront and D_j. The left intersection point s_1 is between the top arc of D_j and an arc a_k of the wavefront, which is part of the bottom arc of a unit disk D_k belonging to a vertex p_k with $i < k < j$. By Lemma 8.2, D_k contains the whole wavefront, thus including s_2, which means D_k and D_j intersect a second time such that this intersection point is to the left of s_2. Then, however, D_k and D_j intersect once with both bottom arcs and once with a bottom and a top arc – a contradiction to Lemma 8.1. For more details on this argument, see the proof of Lemma 8.2 and Figure 8.5.

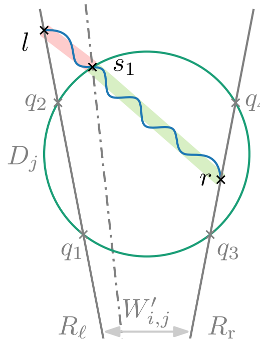
Case TM: $q_1 \prec q_2 \prec l$ and $q_3 \prec r \prec q_4$; see Figure 8.8b. There is an intersection point s_1 between D_j and the wavefront. We traverse[25] the arcs in the balanced search tree representing the wavefront starting at the leftmost arc, which in turn starts at point l, and remove all arcs that we encounter until we find the intersection point s_1 between D_j and an arc a of the wavefront. We update a to start at s_1 and the left bounding ray of $W_{i,j}$ to go through s_1. There cannot be a second intersection point because otherwise there would also be a third intersection point between a unit disk and the wavefront.

Case TT: $q_1 \prec q_2 \prec l$ and $q_3 \prec q_4 \prec r$; see Figure 8.8c. In this case, we either have two or no intersection points between the wavefront and D_j. We traverse the wavefront starting at l and remove all arcs that we encounter and that do not intersect D_j. If we do not find any intersection point but reach r, then there cannot be any further valid shortcut starting at v_i and we abort. Otherwise, we have found s_1 and proceed symmetrically at r to find s_2.

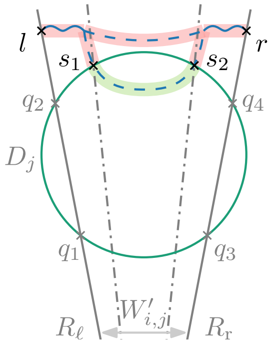
[25] In the following we just say for short, "we traverse the wavefront starting at l".



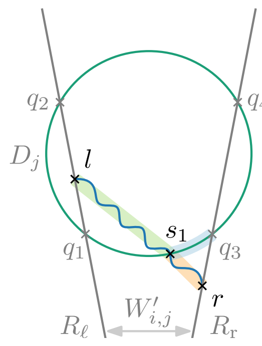
(a) Case TB. (cannot occur)



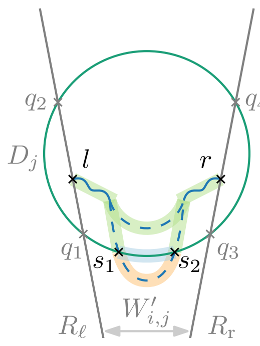
(b) Case TM.



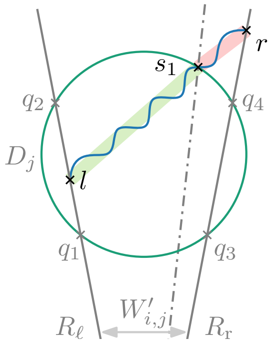
(c) Case TT.



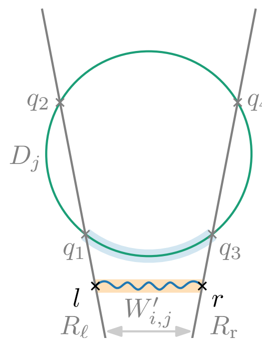
(d) Case MB.



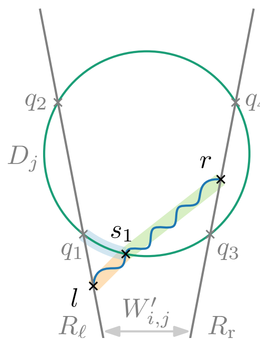
(e) Case MM.



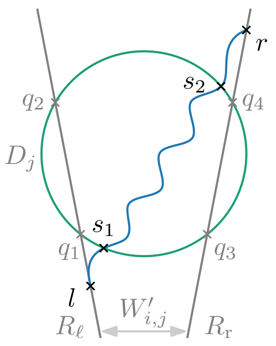
(f) Case MT.



(g) Case BB.



(h) Case BM.



(i) Case BT. (cannot occur)

Figure 8.8: Cases for updating the wavefront when a vertex p_j (with unit disk D_j) is added. The wavefront of the previous step is illustrated as a blue wavy line. Green background color (everything inside D_j) highlights the parts that remain part of the wavefront and red and orange background color (everything outside D_j) highlight the parts that are removed from the wavefront. The part of the bottom arc of D_j that becomes part of the wavefront is highlighted by blue background color.

Case MB: $q_1 \prec l \prec q_2$ and $r \prec q_3 \prec q_4$; see Figure 8.8d. There is precisely one intersection point s_1 between the wavefront and the bottom arc of D_j. We traverse the wavefront starting at r and remove all arcs that we encounter and that do not intersect D_j until we have found s_1. We clip the arc of the wavefront at s_1 and append the bottom arc of D_j between s_1 and r to the wavefront.

Case MM: $q_1 \prec l \prec q_2$ and $q_3 \prec r \prec q_4$; see Figure 8.8e. There are either two or no intersection points between the wavefront and D_j. If there are two intersection points, then they are on the bottom arc of D_j as otherwise, it would contradict Lemma 8.7. The order of the arcs on the wavefront around p_i is reverse to the order of the corresponding unit disk centers around p_i [MO88]. By binary search, we determine the arcs a_k and a_{k+1} such that p_j lies between the corresponding unit disk centers of a_k and a_{k+1} with respect to the angle around p_i. If a_k and a_{k+1} are completely contained inside D_j, then there is no intersection point and the wavefront remains unchanged.

Otherwise, we traverse the wavefront starting at a_k to the left until we have found an arc intersecting D_j, which gives us s_1. We remove all arcs along the way and split the arc containing s_1 at s_1. Symmetrically, we traverse the wavefront starting at a_{k+1} to the right to find s_2. Finally, at the resulting gap, we insert the arc of D_j between s_1 and s_2 into the wavefront.

Case MT: $q_1 \prec l \prec q_2$ and $q_3 \prec q_4 \prec r$; see Figure 8.8f. This case is symmetric to Case TM.

Case BB: $l \prec q_1 \prec q_2$ and $r \prec q_3 \prec q_4$; see Figure 8.8g. There is no intersection point between the wavefront and D_j. There cannot be a single intersection point and if there were two intersection points, it would contradict Lemma 8.7. We replace the whole wavefront by the arc of D_j from q_1 to q_3.

Case BM: $l \prec q_1 \prec q_2$ and $q_3 \prec r \prec q_4$; see Figure 8.8h. This case is symmetric to Case MB.

(Case BT:) $l \prec q_1 \prec q_2$ and $q_3 \prec q_4 \prec r$; see Figure 8.8i. Since this configuration is symmetric to Case TB, this case also cannot occur.

Note that we only do binary search in logarithmic time or if we traverse multiple arcs, we remove them from the wavefront. During the whole process, we add, for any vertex p_j with $j > i$, at most one arc to the wavefront. So, we conclude Lemma 8.11.

Lemma 8.11. *Given a two-dimensional n-vertex polyline L and a vertex $p \in L$, we can find all valid shortcuts under the local Fréchet distance starting at p in $\mathcal{O}(n \log n)$ time.*

This update process in amortized logarithmic time per vertex is tailored specifically for this polyline simplification algorithm. We remark, however, that the more general problem of determining the two, one, or zero intersection points between a unit disk and the wavefront can also be accomplished in logarithmic time by a recursive case distinction.

Now we have all ingredients to prove our main theorem.

Theorem 8.12. *A two-dimensional n-vertex polyline can be simplified optimally under the local Fréchet distance in the L_2-norm (i.e., the Euclidean norm) in $\mathcal{O}(n^2 \log n)$ time and $\mathcal{O}(n)$ space.*

Proof. According to Lemmas 8.5 and 8.6, the algorithm we describe in Section 8.3.1 finds all valid shortcuts. It remains to analyze the runtime. We consider each of the n vertices as potential shortcut starting point p_i. When we encounter a vertex p_j with $j > i$, we determine in logarithmic time whether it is in the valid region. We do this by computing the ray emanating at p_i and going through p_j, and querying the arc it intersects in the wavefront. Then, using the case distinction of Lemma 8.11, we update the wavefront and the wedge. This needs amortized logarithmic time and over all steps $\mathcal{O}(n \log n)$ time.

Consequently, we construct the shortcut graph in $\mathcal{O}(n^2 \log n)$ time. In the resulting shortcut graph, we can find an optimal polyline simplification by finding a shortest path in $\mathcal{O}(n^2)$ time.

Regarding space consumption, we observe that the wavefront maintenance only requires linear space at any time. As we can compute the set of outgoing shortcuts of each vertex p_i individually, we can easily apply the space reduction approach described for the Imai–Iri algorithm in Section 8.2.1 to get an overall space consumption in $\mathcal{O}(n)$. □

8.4.3 Extension to General L_p-Norms

We can use our data structure for the L_2-norm also for any L_p-norm with $p \in (1, \infty)$. However, we should take this with a grain of salt as computing the intersection points between lines and unit disks and between pairs of unit disks in the L_p-norm for $p \in (1, \infty) \setminus \{2\}$ may involve solving equations of degree p, which may raise numerical issues for the required precision. To avoid this, we assume for the following corollary that we can determine the exact intersection points between unit disks and lines (or other unit disks) in every L_p-norm in constant time.

Corollary 8.13. *A two-dimensional n-vertex polyline can be simplified optimally under the local Fréchet distance in the L_p-norm for any $p \in (1, \infty)$ in $\mathcal{O}(n^2 \log n)$ time and $\mathcal{O}(n)$ space, assuming that we can compute the exact intersection points between a unit disk and a line and between two unit disks in constant time.*

8.5 Use Cases with Small Wavefronts

The most complicated part of the algorithm is the efficient maintenance of the wavefront. But in case the wavefront has low complexity, we do not need any dynamic binary search data structure to make updates, but we can simply iterate over a linked list representing the whole wavefront to determine the relevant intersection points and to perform the dynamic changes. We next discuss use cases where the wavefront complexity is provably small.

8.5.1 Simplifying under the L_1- and L_∞-Norms

In the L_1- and L_∞-norms, which are also known as *Manhattan* and *maximum* norm, respectively, the unit disks are square-shaped. Thus, the wavefront consists of a sequence of orthogonal line segments. As we show in the next lemma, this reduces the potential size of the wavefront significantly.

Lemma 8.14. *In the L_1- and L_∞-norms, the wavefront always consists of either one or two (orthogonal) straight line segments. These straight line segments are horizontal or vertical in the L_∞-norm and rotated by 45 degrees in the L_1-norm.*

Proof. We show this claim inductively. For $W_{i,i+1} = O_{i,i+1}$ it is just the bottom arc of a square (which is the shape of a unit disk in the L_1- and L_∞-norms). Clearly, this is either one or two orthogonal line segments – horizontal or vertical line segments in the L_∞-norm and line segments rotated by 45 degrees in the L_1-norm.

When we compute the wavefront of $W_{i,j}$, we compute the intersection of the valid region of $W_{i,j-1}$ (which is bounded by one or two orthogonal line segments by the induction hypothesis) and the local valid region of $O_{i,j}$ (which is bounded by one or two line segments parallel to the ones of $W_{i,j-1}$). Computing the boundary of this intersection in the L_∞-norm can be done by computing the intersection of two axis-parallel rectangles. The intersection of two axis-parallel rectangles is again an axis-parallel rectangle. In the L_1-norm, the situation is the same but rotated by 45 degrees. □

Lemma 8.14 directly yields the following theorem when using the algorithm described in Section 8.3.

Theorem 8.15. *A two-dimensional n-vertex polyline can be simplified optimally under the local Fréchet distance in the L_1- and L_∞-norms in $\mathcal{O}(n^2)$ time and $\mathcal{O}(n)$ space.*

8.5.2 Light Polylines

In Lemma 8.2, we have observed that for any vertex p whose unit disk D contributes to the current wavefront, D actually contains the complete wavefront. Thus, if two vertices contribute an arc to the current wavefront, they are within a distance of 2δ, i.e., inside a unit disk. To end up with a complex wavefront, there hence need to be many vertices in close proximity (and they also need to occur in a specific pattern for all of their unit disks contributing to the wavefront simultaneously). Accordingly, if we consider polylines with bounded vertex density, the wavefront complexity is bounded as well. To formalize this, we introduce the natural class of ν-*light* polylines.

Definition 8.16. A polyline L in d dimensions is ν-*light* if for any $k \in \{2, \dots, n\}$, no set (in particular not the closest set) of k vertices of L lies in a ball of radius less than $(k/\nu)^{1/d}$.

Before we exploit the properties of ν-lightness in the context of polyline simplification, we want to gain some more intuition behind this definition. If a polyline in two dimensions is ν-light, this guarantees that the vertices are somewhat well distributed: the closest pair of vertices has a distance of at least $2 \cdot \sqrt{2/\nu}$, the closest triple of vertices has a surrounding disk of diameter at least $2 \cdot \sqrt{3/\nu}$ and so on. An alternative (and maybe more intuitive) definition is that a polyline L is ν-light if for any point $p \in \mathbb{R}^d$ and any radius $r > 0$, the number of vertices of L inside the ball $B_r(p)$ of radius r centered at p is at most $\max\{\nu r^d, 1\}$. This shows the connection to the related concepts of c-packed curves, ϕ-low density curves, and κ-bounded curves studied in previous work to show improved bounds, e.g., for computing the (approximate) Fréchet distance between two curves [DHW12]. The main difference is that these classifications do not distinguish between polyline vertices and points on the straight line segments in between, which, however, is important in our scenario.

Now let us revisit our algorithm for polyline simplification with distance threshold δ for the local Fréchet distance. In a ball of radius $r = \delta$ (i.e., a unit disk), a two-dimensional ν-light polyline has $\mathcal{O}(\nu\delta^2)$ vertices. Hence, for any constant choice of δ, the wavefront complexity is $\mathcal{O}(\nu)$. The running time of the algorithm is then in $\mathcal{O}(n^2 \log \nu)$, or in $\mathcal{O}(n^2\nu)$ when omitting the tree data structure. So if $\nu \in \mathcal{O}(1)$, the resulting running time is quadratic even without using a dedicated dynamic data structure.

Theorem 8.17. *A two-dimensional ν-light n-vertex polyline can be simplified optimally under the local Fréchet distance in any L_p-norm with $p \in [1, \infty]$ in $\mathcal{O}(n^2)$ time and $\mathcal{O}(n)$ space, given that we can compute intersection points between a unit disk and a line and between two unit disks in constant time.*

8.6 Concluding Remarks and Open Problems

We have identified and closed a seeming gap in literature concerning a natural problem in computational geometry. Namely, the question of whether there is a subcubic-time algorithm that computes for a given polyline an optimal simplification under the local Fréchet distance. Simultaneous to us, Buchin et al. [BvdHO$^+$22] have answered this question positively by providing an $\mathcal{O}(n^{5/2+\varepsilon})$-time algorithm as an application of a new data structure. We have described an algorithm with a running time in $\mathcal{O}(n^2 \log n)$, which is worst-case optimal up to a logarithmic factor for any algorithm that explicitly or implicitly constructs the whole shortcut graph. Our algorithm is simpler and faster than the one of Buchin et al.

Our algorithm does not provide new techniques, but we use, modify, and combine existing approaches for similar polyline simplification algorithms, which have not been described for this precise setting yet. Although the Chan–Chin algorithm for the local Hausdorff distance is asymptotically faster, it requires two sweeps over the polyline to identify the set of valid shortcuts, while we maintain the elegance of the Melkman–O'Rourke algorithm by requiring only a single sweep. Moreover, our algorithm is simpler than the line stabbing algorithm by Guibas et al.

As polyline simplification is a building block, for example, for trajectory clustering algorithms [BBK^{+}20] or simplification algorithms for more generalized structures [BSS^{+}21], our result may also help to improve running times of such approaches. Due to its basic nature, there are many more use cases conceivable where this algorithm can serve as a black-box subroutine. Also, note that the running time of our algorithm is only by a logarithmic factor slower than the widely used Douglas–Peucker heuristic for the local Fréchet distance [vKLW20], but our algorithm computes optimal simplifications.

Moreover, we conjecture that in practice, the running time of the algorithm we describe should be quadratic even if one omits the tree data structure and simply uses a linked list and linear searches to maintain and update the wavefront. For a large wavefront to arise, the polyline vertices need to form a specific pattern, which is unlikely to occur naturally. It would be interesting to validate this claim empirically – maybe even including the concept of ν-light polylines.

Furthermore, the investigation of lower bounds could shed light on the question of whether our upper bounds are tight. Existing lower bounds only apply to simplification of polylines in high dimensions. For the practically most relevant use case of two dimensions, no (conditional) lower bounds are known, though. Another direction for future work would be to generalize the algorithm to work in higher dimensions, where the wavefront becomes more complex. Finally, one could also consider further distance measures, e.g., the Fréchet distance under the L_p-metric for $p \in (0, 1)$, where the respective unit disks are not convex anymore, which could make updating the wavefront data structure more expensive.

In some applications, it can be a drawback that in the classical definition of the polyline simplification problem, the two endpoints of a polyline always need to be kept. One could relax this constraint and investigate the setting that also the endpoints may be removed as long as this removal does not violate the distance constraint. This may be done in a (user) study using real-world examples.

Chapter 9

Consistent Simplification of Polyline Bundles

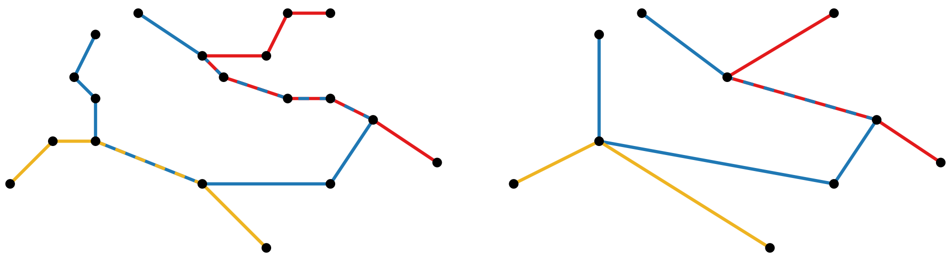
In the previous chapter, we have seen that polylines are a geometric structure with plenty of real-world applications. However, in some applications, we do not just have single polylines, but more objects like points or rectangular labels, which represent cities and toponyms and impose additional topological constraints for polyline simplifications. After simplification, a city of the one country should not appear inside a neighboring country or the sea. Furthermore, we maybe can assume that a coastline is represented by a single polyline, but already for streets, country borders, and rivers, the structure is rather a drawn planar graph. In a straightforward fashion, all we can do there is decomposing such a planar graph to several polylines by splitting all vertices of degree ≥ 3 to endpoints of paths. Afterwards, we can simplify each resulting polyline individually. This has the drawback that these vertices of degree ≥ 3 become fixed points, which may spoil a good simplification.

Now consider a metro map. There, we usually have several public transport lines that are represented as partially overlapping polylines. In this chapter, we illustrate why simplifying each polyline individually may cause problems and we henceforth define rules how to simplify multiple polylines *consistently*. Then, we show that, unlike for single polylines, simplifying multiple polylines consistently is a hard problem, which, however, can be tackled by a specific approximation algorithm.

9.1 Introduction

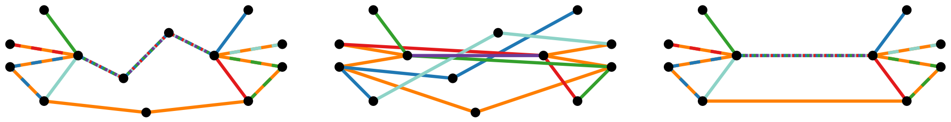
On a map, there are usually multiple polylines to display. Such polylines may share vertices and line segments between vertices sectionwise. We call them a *polyline bundle*. For an illustration, see Figure 9.1. A good example is a schematic map of a public transport network where bus and metro lines are the polylines and these share some of the stations and legs. Other examples are trajectories of cars that are on the same roads for a while and then their paths may split and re-join, or the visualization of a flow network, where elementary flows may share edges and may separate or merge at vertices.

We know how to efficiently simplify a single polyline, so a naive approach would be to simplify the polylines of a bundle independently. This has some drawbacks, though. On the one hand, the total complexity tends to increase when the shared parts are simplified in different ways; see Figure 9.2. On the other hand, it might suggest a misleading picture when we remove common segments and vertices of some



(a) before consistent simplification **(b)** after consistent simplification

Figure 9.1: Consistent simplification of a polyline bundle with three polylines (indicated by color).



(a) initial polyline bundle (15 segments) **(b)** independently simplified polylines (17 segments) **(c)** consistently simplified bundle (12 segments)

Figure 9.2: A polyline bundle with six polylines (indicated by color) that are simplified once independently and once consistently in a bundle. In the independently simplified polyline bundle, the number of vertices remains the same, but the individual polylines are inconsistent in which vertices they keep. This gives an unclearer picture with increased complexity.

polylines, but not of all. The viewer might get the wrong impression that the one route has taken some street or passed through some area and the other has not, while in reality both took the same route in this place. E.g., if there is only one way to pass through some point, say a tunnel, then the simplifications of all polylines going through this point should still share the corresponding vertex or segment if it is kept. Therefore, we require that a vertex in a simplification of a polyline bundle is either kept in all polylines containing it or discarded in all polylines. Note that in Figure 9.1, the given polyline bundle is simplified consistently. The objective is to minimize the total number of vertices that are kept.

Related Work. The so-called *chain pair simplification problem* asks for the simplifications of two given polylines such that, for a given $k \in \mathbb{N}$ and $\delta > 0$, each simplified chain contains at most k segments, and the Fréchet distance between them is at most δ [BJW$^+$08]. The problem arises in protein structure alignment or map-matching tasks and was studied from a theoretical and practical perspective [WZ13, FFK$^+$15, FFKZ16]. While the basic idea to preserve resemblance between polylines after simplification is similar to the motivation behind our problem of polyline bundle simplification, chain pair simplification only ever considers two polylines and does not put further restrictions on the simplification of shared parts.

Analyzing bundles of (potentially overlapping and intersecting) movement trajectories is an important means to study group behavior and to generate maps. For example, the RoadRunner approach [HBA$^+$18] infers high-precision maps from

GPS trajectories. Buchin, Kilgus, and Kölzsch [BKK18] proposed an approach that computes a concise graph that represents all trajectories in a given set sufficiently well. However, these and similar methods do not produce valid simplifications of each input polyline but allow discarding outliers or letting a polyline be represented by a completely disjoint polyline, which is quite different from our setting of polyline bundles. For more related work on map construction, which also uses the Fréchet distance, see the book by Ahmed, Karagiorgou, Pfoser, and Wenk [AKPW15].

There is a multitude of polyline simplification problem variants for single polylines which involve additional constraints. One important variant is the computation of the smallest possible simplification of a single polyline which avoids self-intersection [dBvKS98]. Another practically relevant variant is the consideration of topological constraints. For example, as mentioned before, if a polyline represents a country border, important cities within the country should remain on the same side of the polyline after simplification. It was proven that these problem variants are hard to approximate within a factor of $n^{\frac{1}{5}-\varepsilon}$ [EM01]. Hence, in practice, they are typically tackled with heuristic approaches [EM01, FMM$^+$17].

Note that the only allowed inputs to these problem variants are either a single polyline without self-intersections or a set of polylines without self-intersections and without common vertices or segments (except for common start and end points). In contrast, we explicitly allow non-planar inputs and polyline bundles in which vertices and segments may be shared among multiple polylines. We also remark that the known results on hardness of approximation of these problems heavily rely on the constraint that feasible solutions are still non-intersecting. Since we do not require this, we have to resort to different techniques.

There has also been some research on approximating optimal polyline simplifications. Agarwal et al. [AHMW05] describe an $\mathcal{O}(n \log n)$ time approximation algorithm for (classical) polyline simplification under the Fréchet distance. It is an approximation algorithm in the sense that the output simplification for distance threshold δ has at most as many vertices as an optimal solution with distance threshold $\delta/2$. We later also relate the size of our approximate solution respecting a distance threshold of δ for a polyline bundle to an optimal solution with distance threshold $\delta/2$.

There are two bi-criteria (α, β)-approximation algorithms for weak polyline simplification[26] under the global Fréchet distance known, where α is the approximation factor for the number of retained vertices and β is the approximation factor for an allowed distance threshold violation: a (1, 8)-approximation in $\mathcal{O}(n \log n)$ time [AHMW05] and, for any $\varepsilon > 0$, a (2, 1+ε)-approximation in $\mathcal{O}(n^2 \log n \log \log n)$ time [vdKKL$^+$19].

The (k, λ)*-center (-median) clustering problem for polylines*[27] has been introduced by Driemel, Krivošija, and Sohler in 2016 [DKS16]. Given a set $\mathcal{L}$ of polylines ($|\mathcal{L}| > k$), the problem asks for a set $\mathcal{C}$ of k polylines with λ vertices each, such that the maximum (the sum) of the Fréchet distances between each polyline $L \in \mathcal{L}$ and the

[26] For the definition of *weak polyline simplification* see Chapter 8.

[27] In literature, the term (k, ℓ)*-center (-median) clustering* is used. As we use ℓ already for the number of polylines in a polyline bundle, we use λ here instead to avoid confusion.

polyline in $\mathcal{C}$ being closest to L is minimized. For polylines in two dimensions, there exists a 3-approximation for the (k, λ)-center clustering problem, but, even if $k = 1$, it is NP-hard to approximate within any factor smaller than 2.25 and $W[1]$-hard in the number of polylines [BDG$^+$19, BDS20]. For polylines in d dimensions, there exists a randomized bi-criteria approximation algorithm for the (k, λ)-median clustering problem [BDR23]. It approximates the solution in both, the Fréchet distance $(1 + \varepsilon)$ and, for the polylines in $\mathcal{C}$, the number of vertices $(2\lambda - 2)$. The running time of the algorithm is exponential in d, λ, $1/\varepsilon$, and a parameter for the failure probability. Cheng and Huang [CH23] improve this result by not increasing the number of vertices (i.e., theirs is not a *bi-criteria* approximation algorithm). Also, they provide an approximation algorithm for the following generalized polyline simplification problem. Given a number $\lambda \in \mathbb{N}$ and a set $\mathcal{L}$ of d-dimensional polylines with an individual distance threshold δ_i per polyline $L_i \in \mathcal{L}$, the task is to find a simplified polyline S with λ vertices such that, for each $L_i \in \mathcal{L}$, the Fréchet distance between L_i and S is at most δ_i. As a corollary, they obtain a bi-criteria approximation algorithm that approximates, for a single d-dimensional polyline, an optimal solution in both, the Fréchet distance $(1 + \varepsilon)$ and the number of vertices $(1 + \alpha)$ for fixed $\varepsilon, \alpha > 0$. Although these papers consider multiple polylines, there are quite some differences to polyline bundles. Most notably, their polylines do not explicitly share vertices and segments. Also, in the simplification step, we do not aim for fewer polylines and, consequently, a weak simplification. Instead, for every original polyline, there is a specific simplified polyline whose vertices are a subset of the original polyline.

Contribution. We introduce the problem of polyline bundle simplification in Section 9.2. Roughly speaking, we are given ℓ polylines on an underlying set P of n points that represent the vertices as well as an error bound δ and we seek to find a subset P^* of P such that, for each polyline L, the local Hausdorff or Fréchet distance between L and the simplified version of L where the vertices in $P \setminus P^*$ have been removed is at most δ, and $|P^*|$ is minimized.

While the optimal simplification of a single polyline can be computed in polynomial time, we show in Section 9.3 that polyline bundle simplification is NP-hard to approximate within a factor of $n^{\frac{1}{3}-\varepsilon}$ for any $\varepsilon > 0$. This result applies already to bundles of two polylines, hence excluding an FPT-algorithm depending on parameter ℓ. We extend this hardness of approximation bound also to the case of planar polyline bundles where the line segments can only intersect in their endpoints.

On the positive side, we show in Section 9.4 that this strong inapproximability can be overcome when relaxing the error bound δ slightly. In other words, we design a bi-criteria approximation algorithm. We allow the simplified polylines in our solution to have a Fréchet distance of 2δ instead of only δ to the original polylines. We can then approximate the optimal solution for the original choice of δ within a factor logarithmic in the input size. As the choice of δ for real-world problems often is made in a rather ad-hoc fashion and uncertainties with respect to the precision of the input polylines have to be factored in as well, we deem our bi-criteria approximation to be of high practical relevance.

We show in Section 9.5 that, while the number of polylines in the bundle is not suitable to obtain an FPT-algorithm, the problem of polyline bundle simplification is fixed-parameter tractable in the number of vertices that are shared among the polylines.

9.2 Problem Definition

A *polyline bundle* $\mathcal{L}$ is a set of polylines $\{L_1, \ldots, L_\ell\}$ for some $\ell \geq 2$ that may share common vertices and (where two polylines share two subsequent vertices) line segments.

An instance of the *polyline bundle simplification* problem (from now on abbreviated by PBS) is specified by a triple $(P, \mathcal{L}, \delta)$, where $P = \{p_1, \ldots, p_n\}$ is a set of n points (*vertices*) in the plane, a polyline bundle $\mathcal{L}$ using only vertices from from P, and a distance threshold parameter δ, which specifies a threshold for some distance measure d_X between original and simplified polyline bundle. Each polyline $L_i \in \mathcal{L}$ ($i \in \{1, \ldots, \ell\}$) is simple in the sense that all vertices of L_i are distinct points of P.

Definition 9.1 (Polyline Bundle Simplification (PBS))**.** Given a triple $(P, \mathcal{L}, \delta)$, the objective is to obtain a minimum-size subset $P^\star \subseteq P$ of points, such that for each polyline $L_i \in \mathcal{L}$ its induced simplification S_i (which is $L_i \cap P^\star$ while preserving the order of points)

- contains the start and the end point of L_i, and
- $d_X(L_i, S_i) \leq \delta$, i.e., for each original polyline and its simplification, a distance measure d_X for polylines is at most δ.

Note that taking a subset of the vertices and keeping precisely these vertices in all polylines directly yields a consistent simplification of the polyline bundle. Moreover, note that there always exists a solution to every PBS instance since setting $P^\star = P$ is always possible.

In general, the input and the output polylines may intersect themselves or other polylines. If the input polylines do not intersect themselves or other polylines (apart from common vertices in P), we say the instance is *planar* and we call the problem *planar* PBS. Note that the output polyline in the planar PBS may contain intersections. If we required also the output polyline to be intersection free, the problem would become NP-hard even for one polyline [EM01].

9.3 Hardness of Approximation

In this section, we describe a polynomial-time reduction from the *minimum independent dominating set* problem (MIDS) to (planar) PBS to show NP-hardness and hardness of approximation. The reduction applies to both, the local Hausdorff and the local Fréchet distance used as a distance measure for PBS.

9.3.1 Minimum Independent Dominating Set (MIDS)

In the MIDS problem, we are given a graph $G = (V, E)$, where V is the node[28] set and E is the edge set of G. We define $\hat{n} = |V|$ and $\hat{m} = |E|$. The objective is to find a set $V^\star \subseteq V$ of minimum cardinality[29] that is a dominating set of G as well as an independent set in G. A dominating set contains for each node v, v itself, or at least one of v's neighbors. An independent set contains for each edge at most one of its endpoints. Halldórsson [Hal93] has shown that MIDS, which is also referred to as the *minimum maximal independent set* problem, is NP-hard to approximate within a factor of $|V|^{1-\varepsilon}$ for any $\varepsilon > 0$. In his proof, he uses a reduction from the satisfiability problem (SAT) to MIDS: from a SAT formula Φ, he constructs a graph such that an algorithm approximating MIDS would decide if Φ is satisfiable.

We observe that this reduction preserves the inapproximability gap of $|V|^{1-\varepsilon}$ even if Φ is a 3-SAT formula. Moreover, we observe that the number of edges in the graph constructed in this reduction by a 3-SAT formula is linear in the number of nodes. Thus, we conclude the following corollary and assume henceforth that we reduce only from sparse graph instances of MIDS, in other words, $\hat{m} \leq c\hat{n}$ for some sufficiently large constant c.

Corollary 9.2. MIDS *on graphs of $\hat{n}$ nodes and $\mathcal{O}(\hat{n})$ edges, i.e., sparse graphs, is NP-hard to approximate within a factor of $\hat{n}^{1-\varepsilon}$ for any $\varepsilon > 0$.*

9.3.2 Reduction from MIDS to PBS

Next, we describe how to construct in polynomial time, for a given graph $G = (V, E)$ (instance of MIDS), a specific PBS instance $(P, \mathcal{L}, \delta)$ with n vertices. Afterwards, we show that, if we could find a simplified polyline bundle of $(P, \mathcal{L}, \delta)$ where the number of retained vertices is at most $n^{\frac{1}{3}-\varepsilon}$ times the number of retained vertices in an optimal simplification for some $\varepsilon > 0$, then we could approximate MIDS within a factor of $n^{1-\varepsilon}$. We analyze the construction with respect to the Hausdorff distance, but observe that the arguments apply to the Fréchet distance as well.

In our construction, every node, every edge, and every neighborhood gets a separate polyline.[30] Hence, we have three types of polylines (*gadgets*) with specific properties. Each of our gadgets looks like a lengthy zigzag piece where shortcuts exist that skip almost all[31] of the vertices of the gadget. Skipping almost all vertices of a gadget can be interpreted as follows.

- In a *node gadget*, it means that a node is **not** in the set V',
- in an *edge gadget*, it means that the independent set property is observed, and
- in a *neighborhood gadget*, it means that the dominating set property is observed.

[28] For a graph in this chapter, we use the term *nodes* instead of *vertices* to distinguish them from the vertices of a polyline.

[29] Below, we use V' to denote some, not necessarily minimum, independent dominating set in G.

[30] These polylines can then be connected to have only two polylines in the bundle; see Section 9.3.6.

[31] We describe below what *almost all* means. Essentially, there is a large gap between skipping only a few and almost all vertices of a gadget.

The node gadgets extend vertically and they are arranged next to each other in some arbitrary order from left to right. The edge and neighborhood gadgets extend horizontally and they share a vertex with each node gadgets they correspond to.[32] Our construction is illustrated in Figure 9.3. Figures 9.3a, 9.3c and 9.3d show the individual gadgets and Figure 9.3e shows an example of how they look combined.

All gadgets are explained in detail next. We define our gadgets with respect to the distance threshold δ, which can be chosen arbitrarily. In our formulas, we also use some $\gamma \leq 2\delta/(30c\hat{n}^2 + 5)$, the constant $c = \hat{m}/\hat{n}$ (and w.l.o.g. $c \geq 1$), and for the horizontal distance of our node gadgets, we use some $x_{\text{spacing}} \geq (6c\hat{n}^2 + 2)3\delta$. When we speak of *shortcuts in a gadget*, we mean the valid shortcuts that would exist if we consider this gadget as a polyline on its own simplified with distance threshold δ.

Note that our problem definition allows intersections and overlaps of different polylines without having a common vertex or segment (*non-planar input*). In this reduction, there can be such intersections, which, however, do not affect the involved polylines locally. In Section 9.3.3, we describe how to get rid of these intersections (*planar input*).

Node Gadget. For each node, we construct a *node gadget* (see Figure 9.3a). We arrange all node gadgets next to each other on a horizontal line in arbitrary order and with distance x_{spacing} between one and the next node gadget.

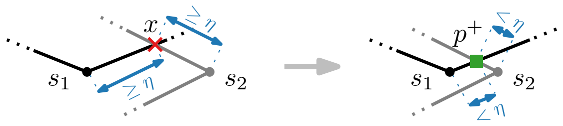
A node gadget has $3\hat{n}(c+1)+2$ vertices arranged in a vertically-stretching zigzag course with x-distance 2δ (δ for the first and the last segment) between each two consecutive vertices. Each third vertex is a shared vertex, so there are $\hat{n}(c+1)$ shared vertices. The y-distance between a non-shared vertex and a shared vertex is $3/5\delta - \gamma$ and $\delta + \gamma$ in the upper part (i.e., the $3c\hat{n} + 1$ topmost vertices of the gadget), and $4/5\delta$ in the lower part (i.e., the $3\hat{n} + 1$ bottommost vertices of the gadget). This y-distance depends on whether the vertex is shared with an edge gadget (upper part) or a neighborhood gadget (lower part). We set the y-distance between each two neighboring non-shared vertices to 3δ.

Claim 9.3. In a node gadget, there is precisely one shortcut, which starts at the first and ends at the last vertex.

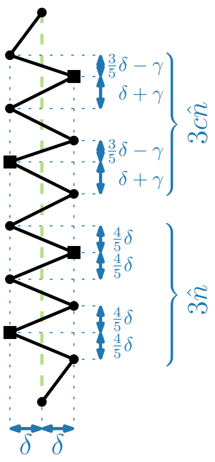
Proof. Clearly, the line segment s from the first to the last vertex has a distance of at most δ to the other vertices and segments of the node gadget, so this shortcut is valid.

It remains to show that there is no other shortcut. Any other potential shortcut segment would cross s at most once. Let this crossing be at a point o. We can assign vertices to the left of s only to the part of the shortcut segment above o and points to the right side of s only to the part of the shortcut segment below o – or the other way around. In both cases, when we traverse the zigzag piece, say, bottom-up, we encounter the vertices alternately on the left and on the right side of s, while the y-coordinates of consecutive vertices are strictly increasing. Thus, there cannot be another shortcut. □

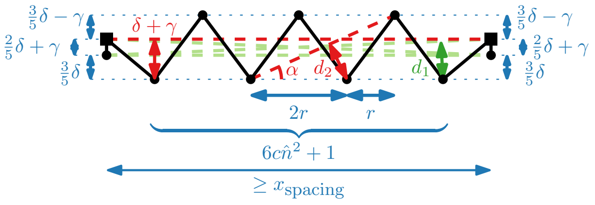
[32] An edge naturally corresponds to two nodes and a neighborhood corresponds to a set of nodes consisting of a node and all of its neighboring nodes.



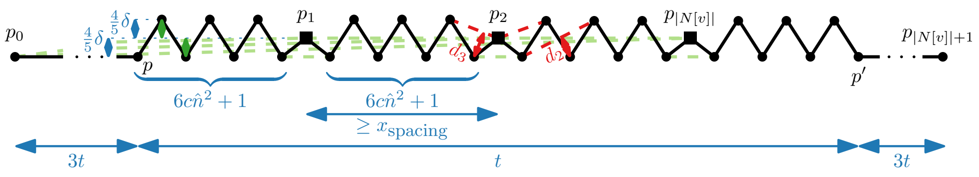
(b) Planarizing a polyline intersection: for an intersection point x, consider two adjacent vertices s_1 (on an edge or neighborhood gadget (black)) and s_2 (on a node gadget (gray)) with the same y-coordinate. Move s_1 towards s_2 until the distance between s_1 (s_2, resp.) and the intersection point is less than η. Then, insert a new vertex p^+ (green square) onto this intersection point.



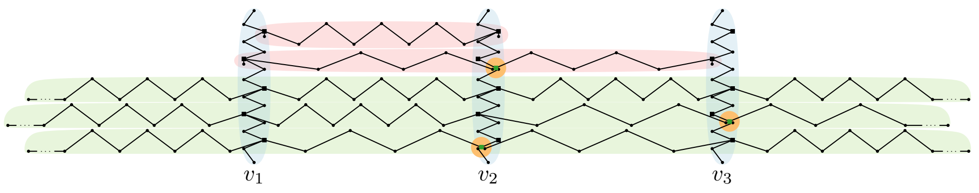
(a) Node gadget.


$\frac{3}{5}\delta - \gamma$
$\frac{2}{5}\delta + \gamma$
$\frac{3}{5}\delta$
$\delta + \gamma$
α
d_2
d_1
$2r$
r
$6c\hat{n}^2 + 1$
$\geq x_{\text{spacing}}$

(c) Edge gadget.



(d) Neighborhood gadget.



(e) Combination of three node gadgets (for the nodes v_1, v_2, v_3; blue background) with two edge gadgets (for the edges v_1v_2 and v_1v_3; red background) and three neighborhood gadgets (for the nodes v_1, v_2, v_3; green background). Planarized crossings are highlighted by orange disks.

Figure 9.3: Schematization of our reduction from MIDS to planar PBS. Nodes shared by two gadgets are drawn as squares. Shortcuts are indicated by dashed green line segments. Dashed red line segments between two vertices indicate that there is no shortcut. The nodes in our minimum independent dominating set are precisely the ones for which we do not take the shortcut of the corresponding node gadgets.

We say that a node v is in V' if and only if we do not skip the inner vertices of the node gadget of v.

Edge Gadget. For each edge uv, we construct an *edge gadget* (see Figure 9.3c) following a horizontally-stretching zigzag piece with $6c\hat{n}^2 + 5$ vertices and sharing its second and second last vertex with one of the two corresponding node gadgets – the node gadgets of u and v. All neighboring vertices from the second to the second last are equidistant in x-dimension, while the first and second vertex, and the second last and last vertex have the same x-coordinate. In y-dimension, the first and the last vertex are $2/5\delta + \gamma$ below the second and second last vertex, respectively. The other vertices are $3/5\delta - \gamma$ above the second vertex or $3/5\delta$ below the first vertex.

Claim 9.4. In an edge gadget, there are precisely three long shortcuts. These are going (i) from the first to the last vertex, (ii) from the first to the second last vertex, and (iii) from the second to the last vertex. (iv) Besides these three (long) shortcuts, there are at most four more (short) shortcuts, which skip only the second and the second last vertex (and possibly also the third and third last vertex). (v) There is no shortcut not skipping one of the shared vertices, i.e., the second or the second last vertex.

Proof. (i) The line segment from the first and to the last vertex is horizontal and has y-distance $3/5\delta$ or $2/5\delta + \gamma$ or δ to all inner vertices. (ii) Regarding a shortcut segment from the first to the second last vertex, it is easy to see that the most critical part is the distance d_1 to the third last vertex. It is the y-distance between the third and the second last vertex, which is $3/5\delta + (2/5\delta + \gamma)$, minus at least a $(6c\hat{n}^2 + 2)$-th of the y-distance between the first and second last vertex, which is $2/5\delta + \gamma$. Combining these values yields

$$d_1 \leq \frac{3}{5}\delta + \left(\frac{2}{5}\delta + \gamma\right) - \frac{\frac{2}{5}\delta + \gamma}{6c\hat{n}^2 + 2} \leq \delta + \gamma - \frac{\frac{2}{5} \cdot \frac{30c\hat{n}^2+5}{2}\gamma + \gamma}{6c\hat{n}^2 + 2} = \delta\,.$$

Clearly, (iii) and (ii) are symmetric.

(v) If neither the second nor the second last vertex is skipped, then we cannot take any shortcut in this gadget. Clearly, we cannot take a "long" shortcut from the second to the second last vertex because the lower row of inner vertices has distance $\delta + \gamma$ from the potential shortcut segment.

Moreover, we cannot take a "short" shortcut from a vertex of the lower row to a non-neighboring vertex of the upper row or the other way around. Assume for a contradiction, we could skip two inner vertices. Then, the distance d_2 (see Figure 9.3c) from an inner vertex to the shortcut segment is at most δ. However, it is

$$d_2 = 2r \cdot \sin\alpha = 2r \cdot \sin\left(\arctan\frac{\frac{8}{5}\delta}{3r}\right) = 2r \cdot \frac{\frac{8\delta}{15r}}{\sqrt{\left(\frac{8\delta}{15r}\right)^2 + 1}} = \frac{16\delta r}{\sqrt{(8\delta)^2 + (15r)^2}}\,, \quad (9.1)$$

where r is the x-distance between two consecutive (inner) vertices. By construction,

$$r \geq \frac{x_{\text{spacing}}}{6c\hat{n}^2 + 2} \geq \frac{(6c\hat{n}^2 + 2)3\delta}{6c\hat{n}^2 + 2} = 3\delta\,, \quad (9.2)$$

and, hence,

$$d_2 \geq \frac{48\delta^2}{\sqrt{64\delta^2 + 2025\delta^2}} = \frac{48}{\sqrt{2089}}\delta = 1.0502\ldots\delta\,. \tag{9.3}$$

Observe that this becomes even greater if we want to skip four or more vertices instead of two vertices. Also, it becomes greater if we start or end at one of the two shared vertices.

(iv) It remains to consider potential shortcuts starting or ending at the first or the last vertex. Clearly, skipping only the second or second last vertex is always possible. Skipping the second and the third vertex or skipping the second last and the third last vertex may sometimes be possible depending on how much the edge gadget is stretched horizontally. However, according to the previous analysis, skipping more vertices is not possible since the distance between the potential shortcut segment and the vertex before the endpoint of the potential shortcut is at least d_2. □

It follows that not skipping one of the two shared vertices is a relatively expensive choice in terms of retained vertices. Remember that not skipping one of the shared vertices means not taking the shortcut in the corresponding node gadget, which means putting the corresponding node into V'. So, skipping almost all vertices in the edge gadget of uv implies not having u or v in V', which means respecting the independent set property for the edge uv.

Neighborhood Gadget. For each node v, we construct a *neighborhood gadget* (see Figure 9.3d). This gadget shares a vertex with every node gadget corresponding to a node of the closed neighborhood $N[v]$ such that these shared vertices have the same y-coordinate.

The node gadgets of $N[v]$ appear in an arbitrary horizontal order in our construction. Say the corresponding nodes in order are $\langle u_1, \ldots, u_{|N[v]|}\rangle$. Let the shared vertices with u_1 and $u_{|N[v]|}$ be p_1 and $p_{|N[v]|}$, respectively. We add a vertex p, which is placed x_{spacing} to the left and $4/5\delta$ below p_1. Symmetrically, we add a vertex p', which is placed x_{spacing} to the right and $4/5\delta$ below $p_{|N[v]|}$. The vertices p and p' are the second and second last vertex of the neighborhood gadget. We place the first vertex of the neighborhood gadget, which we denote by p_0, on the same height and $3t$ to the left of p, where t is the distance between p and p'. Symmetrically, we place the last vertex of the gadget, which we denote by $p_{|N[v]|+1}$, on the same height and $3t$ to the right of p'.

Between each two vertices p_i and p_{i+1} with $i \in \{0, \ldots, |N[v]|\}$, we add a regular horizontally-stretching zigzag piece with $6c\hat{n}^2 + 1$ vertices (including p and p', excluding all p_i). The one half of the vertices of the zigzags is on the same height as p and p' and the other half is $8/5\delta$ above.

Claim 9.5. *In a neighborhood gadget, the only shortcuts (i) skip only p_i with $i \in \{1, \ldots, |N[v]|\}$ or (ii) start at p_j with $j \in \{0, \ldots, |N[v]|\}$ and end at p_k with $k \in \{j+1, \ldots, |N[v]|+1\}$ except for the case that $j = 0$ while $k = |N[v]|+1$.*

Proof. (i) Clearly, for each $i \in \{1, \ldots, |\mathrm{Adj}(v)|\}$, the shortcut that starts at the vertex directly before p_i, skips only p_i, and ends at the vertex directly after p_i is valid.

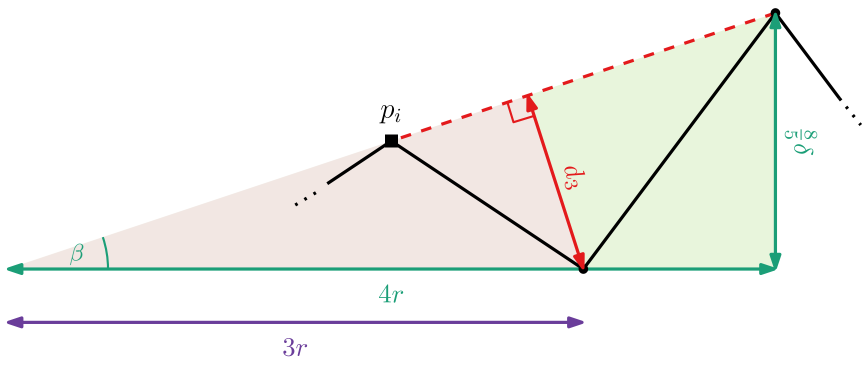


Figure 9.4: The potential shortcut segment in a neighborhood gadget form a vertex p_i to an inner vertex is dashed in red. However, due to $d_3 > \delta$, it is no valid shortcut segment.

(ii) For each $j \in \{1, \ldots, |N[v]|-1\}$ and each $k \in \{j+1, \ldots, |N[v]|\}$, there clearly is a valid shortcut from p_j to p_k. For $j = 0$ and each $k \in \{1, \ldots, |N[v]|\}$, observe that, in the most extreme case, the line segment s from p_0 to $p_{|N[v]|}$ has a y-distance to the upper row of vertices of

$$\frac{4}{5}\delta + \frac{t}{4t} \cdot \frac{4}{5}\delta = \delta$$

when s passes p in x-dimension. Thus, this shortcut is valid and this also holds for $k < |N[v]|$. For each $j \in \{1, \ldots, |N[v]|\}$ and $k = |N[v]| + 1$, this argument applies symmetrically. Obviously, there is no shortcut from p_0 to $p_{|N[v]|+1}$ since the potential shortcut segment has distance $8/5\delta$ to the upper row of vertices.

It remains to argue that there are no more valid shortcuts. A shortcut starting and ending at a vertex on the upper or lower row is not possible because it would either be a horizontal segment, which has distance $8/5\delta$ to the other row, or the distance to some vertex in between would be at least d_2, which we have shown to be greater than δ in Equations (9.1) and (9.3). It is easy to see that there is no shortcut starting at p_0 and ending at some inner vertex of the upper or lower row. The same holds true for shortcuts starting at some inner vertex of the upper or lower row and ending at $p_{|N[v]|+1}$.

Moreover, a shortcut segment starting (ending) at some p_i for $i \in \{1, \ldots, |N[v]|\}$ and skipping one vertex would have a distance of d_3 to this vertex as depicted in Figure 9.4. Since d_3 is inside a rectangular triangle, we can determine d_3 by

$$d_3 = 3r \cdot \sin\beta\,,$$

where r is the x-distance between two consecutive (inner) vertices in the corresponding zigzag piece and β is an angle in another rectangular triangle and thus can be determined by

$$\beta = \arctan\frac{\frac{8}{5}\delta}{4r} = \arctan\frac{2}{5r'}\,,$$

where $r' = r/\delta$. Putting them together, we get

$$d_3 = 3r'\delta \cdot \sin\left(\arctan\frac{2}{5r'}\right) = 3r'\delta \cdot \frac{\frac{2}{5r'}}{\sqrt{1-\frac{4}{25r'^2}}} = \frac{6}{\sqrt{25-\frac{4}{r'^2}}}\delta\,.$$

Since $r' \geq 3$ (see Equation (9.2)), this means $d_3 \geq 1.2108\ldots\delta$.

If we skip more than one inner vertex, the distance to the last skipped vertex becomes even greater than d_3. Hence, we conclude that the claim is correct. □

Due to Claim 9.5, we can skip almost all vertices in a neighborhood gadget if we keep at least one vertex from $\{p_1, \ldots, p_{|N[v]|}\}$, which are the vertices shared with the node gadgets. If we skip all of them, we can skip no other vertex. So, to avoid high costs in terms of retained vertices, we must not take the shortcut of the node gadget of at least one node in $N[v]$. This means that we must, for each $v \in V$, add a node of $N[v]$ to V', which enforces the dominating set property.

9.3.3 Making the PBS Instance Planar

The current construction is (to a high degree) non-planar. We next describe how to make it planar, i.e., polylines cross each other only in common vertices. The key idea is to planarize the non-planar construction by replacing polyline intersections by new vertices, which we call *crossing vertices*. However, we need to be careful where to insert crossing vertices. Just inserting vertices wherever an intersection point occurs could give rise to new shortcuts and hence destroy the mechanics of the gadgets. We can prevent this from happening if we ensure that crossing vertices lie sufficiently close to existing vertices. Then, we cannot gain anything by ending a shortcut at a crossing vertex rather than an original vertex.

First we increase the horizontal distance between each two neighboring node gadgets by a factor of 2. Now, every zigzag piece in every edge gadget and every neighborhood gadget (see Figures 9.3c and 9.3d) has width at least $2x_{\text{spacing}}$ instead of "just" x_{spacing}. This stretches all edge and neighborhood gadgets horizontally and gives us enough flexibility to move each individual inner vertex of these zigzag pieces (up to $0.5r$) to the left or to the right, while maintaining the functionality of the gadgets. Observe that it is a crucial property of our hardness construction that stretching edge and neighborhood gadgets horizontally does not change the behavior in terms of possible shortcuts because we have already assumed that x_{spacing} is only a lower bound for the width; see Claims 9.4 and 9.5.

Now consider an intersection point x (in the stretched drawing) and its two closest non-shared vertices s_1 and s_2, where s_1 lies in a zigzag piece of an edge or a neighborhood gadget and s_2 lies in a node gadget; see Figure 9.3b on the left. First note that s_1 and s_2 share a common y-coordinate by construction (the y-coordinates of the vertices in the upper part of a node gadget coincide with the y-coordinates of the vertices in the edge gadgets, and in the lower part they coincide with the neighborhood gadgets). We move s_1 horizontally towards s_2 such that the distance of the (also moving) intersection point to both s_1 and s_2 is less than η, which we specify below. Onto this carefully arranged intersection point, we now insert a new crossing vertex p^+ to planarize the construction; see Figure 9.3b on the right. Now, p^+ is a vertex of both involved gadgets.

For the correctness of Claims 9.4 and 9.5, we require that the horizontal distance between each two vertices in a zigzag piece of an edge or neighborhood gadget is $\geq 3\delta$; see Equation (9.2). Since we have increased this horizontal distance to $\geq 6\delta$ by horizontal stretching with factor 2 and we have moved s_1 by at most $0.5 \cdot 6\delta$, the horizontal distance of s_1 to its neighbors within the zigzag piece is still $\geq 3\delta$.

We now analyze how close to s_1 and s_2, the crossing vertex p^+ needs to be placed. We require s_1 and s_2 to be strictly inside a disk of radius η around p^+ to prevent the emergence of new shortcuts. Intuitively, η is chosen sufficiently small to ensure that, given any pair of vertices $\langle p, q\rangle$ that do not admit a valid shortcut, moving p or q within a disk of radius η does not bring the line segment $\langle p, q\rangle$ into the radius-δ neighborhood disk of some third vertex o. More formally, we let

$$\eta = \left(\min_{\substack{\{p,o,q\} \subseteq L, \\ L \in \mathcal{L}}} \{ d(\overline{pq}, o) \colon d(\overline{pq}, o) > \delta \} \right) - \delta \,.$$

Observe that we can determine η in polynomial time.

By Lemma 9.6, we show that the new crossing vertices do not allow new shortcuts and, hence, the functionality of the gadgets is not affected regardless of whether we keep the crossing vertex and skip the neighboring original vertices, which we call its *skip vertices*, or the other way around. For a set of shortcuts Z, we let $\mathsf{P}(Z)$ denote the set of endpoints of all shortcuts in Z.

Lemma 9.6. *Let p^+ be a crossing vertex, and let Z_{p^+}, Z_{s_1}, and Z_{s_2} be the set of shortcuts having p^+ and p^+'s two skip vertices s_1 and s_2 as an endpoint, respectively. Then, $\mathsf{P}(Z_{p^+}) \setminus \{p^+\} \subseteq \mathsf{P}(Z_{s_1}) \cup \mathsf{P}(Z_{s_2})$.*

Proof. We prove this statement by contradiction. Suppose there is a vertex q such that the line segment $\langle p^+, q\rangle$ is a shortcut, whereas $\langle s_1, q\rangle$ and $\langle s_2, q\rangle$ are no shortcuts. W.l.o.g., let p^+, q, s_1 be vertices of the same polyline L. We know that $d_{\mathrm{H}}(\langle p^+, q\rangle, L[p^+, q]) \leq \delta$.

For all of the gadgets, it has been shown that wherever there is no shortcut between two vertices p and q, this is because some vertex o between p and q has Euclidean distance greater than δ to $\overline{pq}$; see Claims 9.3 to 9.5. Hence in our case and by the choice of η, there is a vertex o on $L[s_1, q]$ (and thus also on $L[p^+, q]$) with $d(\overline{s_1 q}, o) \geq \delta + \eta$.

By the choice of p^+, we know that $d_{\mathrm{H}}(\langle p^+, q\rangle, \langle s_1, q\rangle) < \eta$. For any vertex o' on $L[p^+, q]$, this implies, by using the triangle inequality, $d(\overline{s_1 q}, o') < \delta + \eta$. This is a contradiction to the choice of o. $\square$

Also note that we can always skip a crossing vertex as it lies on the line segment between its predecessor vertex and successor vertex on both of its polylines. Hence, we do not count crossing vertices in Section 9.3.5.

9.3.4 Size of the PBS Instance

Observe that all shared vertices are shared between only two polylines – by a node gadget and either an edge gadget or a neighborhood gadget. A node gadget provides enough vertices that may be shared with the edge and neighborhood gadgets as a node is contained in at most $\hat{n}$ neighborhoods and there are at most $c\hat{n}$ edges. In the following lemma, we analyze the size of the constructed planar PBS instance.

Lemma 9.7. *By our reduction, we obtain from an instance $G = (V, E)$ of* MIDS *an instance of* PBS *with a planar polyline bundle that has $n < 50c^2\hat{n}^3$ vertices, where $\hat{n} = |V| \geq 2$, $|E| = c\hat{n}$ ($c \geq 1$ is constant).*

Proof. By construction, we have at most one shared vertex for each pair of node gadget and edge gadget, and for each pair of node gadget and neighborhood gadget. This is a shared vertex either because the corresponding node is incident to the corresponding edge or part of the corresponding neighborhood, or because it is a crossing vertex. So in total we have at most $\hat{n} \cdot (\hat{m} + \hat{n})$ shared vertices. All node gadgets together have $\hat{n}(3c\hat{n} + 3\hat{n} + 2)$ vertices.

For the edge and neighborhood gadgets, we count only non-shared vertices because in the following sum, we add the shared vertices separately. For the node gadgets, we have counted all vertices because not all of its (potentially shared) vertices need to be shared. All edge gadgets have $\hat{m}(6c\hat{n}^2 + 3)$ non-shared vertices, and all neighborhood gadgets have $(2\hat{m} + 2\hat{n}) \cdot (6c\hat{n}^2 + 1) + 2\hat{n}$ non-shared vertices. Summing these values up and using $\hat{m} = c\hat{n}$ yields

$$\begin{aligned} n &\leq \hat{n} \cdot (\hat{m} + \hat{n}) + \hat{n}(3c\hat{n} + 3\hat{n} + 2) + \hat{m}(6c\hat{n}^2 + 3) + (2\hat{m} + 2\hat{n}) \cdot (6c\hat{n}^2 + 1) + 2\hat{n} \\ &= (18c^2 + 12c)\hat{n}^3 + (4c + 4)\hat{n}^2 + (5c + 6)\hat{n} < 50c^2\hat{n}^3 \,. \end{aligned}$$ □

9.3.5 Correctness

We say a simplification of an instance of PBS obtained by this reduction *corresponds* to an independent and dominating set V' and vice versa if we take the (unique) shortcuts in the node gadgets except for the ones corresponding to V' and we skip all inner non-shared vertices (the zigzag pieces) in all edge and neighborhood gadgets, which is possible since V' is independent and dominating. Observe that for each independent and dominating set, there is precisely one corresponding simplification (which is also valid according to δ).

Lemma 9.8. *Let V' be a solution for an instance $G = (V, E)$ of* MIDS. *In the instance $(P, \mathcal{L}, \delta)$ of* PBS *obtained by our reduction, the size of the simplification corresponding to V' is $\hat{n}(3(c + 1)|V'| + 2c + 4)$, where $\hat{n} = |V|$ and $c \geq 1$ is constant.*

Proof. For all vertices except for the ones in V', we take the shortcuts in the corresponding node gadgets in $(P, \mathcal{L}, \delta)$, which reduces the number of vertices in each of these gadgets to 2. This gives us $(\hat{n} - |V'|) \cdot 2 + |V'| \cdot (3\hat{n}(c + 1) + 2) = \hat{n}\,(2 + 3(c + 1)|V'|)$ remaining vertices in all node gadgets combined. In the following, we count shared vertices for the node gadgets. We take a "long" shortcut in all of the edge gadgets. This gives us two remaining non-shared vertices in all edges gadgets ($2c\hat{n}$ vertices in total). Moreover, we skip all inner non-shared vertices in all of the neighborhood gadgets ($2\hat{n}$ vertices remaining). Altogether, this sums up to $\hat{n}(3(c+1)|V'|+2c+4)$. □

By Lemma 9.8, we know that for an optimal solution $V^\star$ of an instance of MIDS, the corresponding simplification in the instance $(P,\mathcal{L},\delta)$ of PBS obtained by our reduction has size $\hat{n}(3(c+1)\mathsf{OPT}_{\text{MIDS}}+2c+4)$, where $\mathsf{OPT}_{\text{MIDS}} = |V^\star|$ and which of course is at least the size $\mathsf{OPT}_{\text{PBS}}$ of the optimal solution of $(P,\mathcal{L},\delta)$. We formalize this in the following lemma.

Lemma 9.9. *For an instance $G=(V,E)$ of* MIDS *and the instance $(P,\mathcal{L},\delta)$ of* PBS *obtained by our reduction from G, $\mathsf{OPT}_{\text{PBS}} \leq \hat{n}(3(c+1)\mathsf{OPT}_{\text{MIDS}}+2c+4)$.*

Theorem 9.10. PBS *with a planar polyline bundle as input is NP-hard to approximate within a factor of $n^{\frac{1}{3}-\varepsilon}$ for any $\varepsilon > 0$, where n is the number of vertices in the polyline bundle. This hardness applies to the local Hausdorff and the local Fréchet distance used as a distance measure for* PBS.

Proof. Suppose for a contradiction that there is an approximation algorithm $\mathcal{A}$ solving any instance of PBS within a factor of $n^{\frac{1}{3}-\varepsilon}$ for some constant $\varepsilon > 0$ with respect to the optimal solution. We can transform any instance $G=(V,E)$ of MIDS, where $\hat{n}=|V|, \hat{m}=|E|$, and $\mathsf{OPT}_{\text{MIDS}} = |V^\star|$, to an instance $(P,\mathcal{L},\delta)$ of PBS using the reduction described above, where $|P|=n$ and the size of an optimal solution is $\mathsf{OPT}_{\text{PBS}}$. This reduction clearly applies to both, the local Hausdorff and the local Fréchet distance.

Employing $\mathcal{A}$ to solve $(P,\mathcal{L},\delta)$ yields a (simplified) polyline bundle $\mathcal{S}_\mathcal{A}$. We denote the number of vertices in $\mathcal{S}_\mathcal{A}$ by $n_\mathcal{A}$ and we know that $n_\mathcal{A} \leq \mathsf{OPT}_{\text{PBS}} \cdot n^{\frac{1}{3}-\varepsilon}$ for some $\varepsilon > 0$. Suppose in $\mathcal{S}_\mathcal{A}$, there was a $(6c\hat{n}^2+1)$-vertex zigzag piece of an edge or neighborhood gadget (see Figures 9.3c and 9.3d) that was not skipped. Then, because the two end vertices of every gadget can also not be skipped and there are $\hat{n}+c\hat{n}+\hat{n}$ gadget in total, it would be $n_\mathcal{A} \geq 6c\hat{n}^2+1+2(\hat{n}+c\hat{n}+\hat{n}) > 6c\hat{n}^2+(2c+4)\hat{n}$. However, since there exists some independent dominating set for G containing at most $\hat{n}$ nodes, there exists a simplification of size at most $\hat{n}(3(c+1)\hat{n}+2c+4) \leq 6c\hat{n}^2+(2c+4)\hat{n}$ due to Lemma 9.8 (recall that $c \geq 1$). Hence, we can assume that all zigzag pieces of the edge and neighborhood gadgets are skipped in $\mathcal{S}_\mathcal{A}$ (otherwise we can replace $\mathcal{S}_\mathcal{A}$ by a solution with fewer vertices by finding any independent dominating set of G greedily in polynomial time), and therefore, we can immediately read an independent dominating node set $V' \subseteq V$ from the node gadgets where the shortcut is not taken.

Using our assumption together with Lemma 9.8 and Lemma 9.9, we get

$$n^{\frac{1}{3}-\varepsilon} \geq \frac{n_\mathcal{A}}{\mathsf{OPT}_{\text{PBS}}} \geq \frac{\hat{n}(3(c+1)|V'|+2c+4)}{\hat{n}(3(c+1)\mathsf{OPT}_{\text{MIDS}}+2c+4)} > \frac{|V'|}{\mathsf{OPT}_{\text{MIDS}}+\frac{2c+4}{3(c+1)}},$$

which we can reformulate as $|V'| < n^{\frac{1}{3}-\varepsilon}(\mathsf{OPT}_{\text{MIDS}}+\frac{2c+4}{3(c+1)})$. We assume that $\mathsf{OPT}_{\text{MIDS}} > \frac{2c+4}{3(c+1)}$ as otherwise we could check all subsets of V of size at most $\frac{2c+4}{3(c+1)}$

in polynomial time. Similarly, we can assume that $\hat{n}$ is sufficiently large to satisfy $\hat{n}^{2\varepsilon} > 100c^2$. Beside this, we apply Lemma 9.7 and obtain

$$\begin{aligned}|V'| &< n^{\frac{1}{3}-\varepsilon} \cdot 2 \cdot \mathsf{OPT}_{\text{MIDS}} < 2 \cdot (50c^2\hat{n}^3)^{\frac{1}{3}-\varepsilon} \cdot \mathsf{OPT}_{\text{MIDS}} \\ &< 100c^2 \cdot \hat{n}^{1-3\varepsilon} \cdot \mathsf{OPT}_{\text{MIDS}} < \hat{n}^{2\varepsilon} \cdot \hat{n}^{1-3\varepsilon} \cdot \mathsf{OPT}_{\text{MIDS}} = \hat{n}^{1-\varepsilon} \cdot \mathsf{OPT}_{\text{MIDS}} .\end{aligned}$$

Since MIDS is NP-hard to approximate within a factor of $\hat{n}^{1-\varepsilon}$ for any $\varepsilon > 0$, it follows that it is NP-hard to approximate PBS within a factor of $n^{\frac{1}{3}-\varepsilon}$. □

9.3.6 Using only Two Polylines

Currently, we use one polyline per gadget. So, our reduction uses $(2+c)\hat{n}$ polylines in total. We can reduce the number of polylines to two by connecting all node gadgets from left to right in a row (alternating with the connecting segments between bottom and top side), which gives us the first polyline, and by connecting all edge and neighborhood gadgets similarly, which gives us the second polyline. For the latter, directly adding connecting segments between these gadgets can be problematic because new shortcuts or crossings may be created when the horizontal span of two such gadgets is very different and when their end vertices lie between node gadgets.

The neighborhood gadgets are already relatively long and might reach to the left of the leftmost node gadget and to the right of the rightmost node gadget. If not, we can simply stretch the two outermost zigzag pieces horizontally without violating the functionality of the gadgets (see Section 9.3.3). Then, we can simply connect the endpoints of the neighborhood gadgets without creating new shortcuts.

The edge gadgets, however, have their endpoints between the node gadgets. The solution is to extend them to reach to the left and the right of all node gadgets similar to the neighborhood gadgets. There, we connect them without creating new crossings or shortcuts. As for the neighborhood gadgets, we do this by adding two additional zigzag pieces – one before the first and one after the last vertex of the edge gadget, which cross all node gadgets to the left and right. This does not violate the functionality of the edge gadget (in particular, consider the case that the two shared vertices of an edge gadget are kept). Observe that this also does not affect the approximation ratio asymptotically. Overall, we conclude the following corollary.

Corollary 9.11. PBS *is not fixed-parameter tractable (FPT) in the number of polylines ℓ. In particular,* PBS *with two polylines is already NP-hard to approximate within a factor of $n^{\frac{1}{3}-\varepsilon}$ for any $\varepsilon > 0$. This holds true even for planar polyline bundles and for both the local Hausdorff and the local Fréchet distance.*

9.4 Bi-criteria Approximation

In this section, we describe a bi-criteria approximation algorithm for PBS under the local Fréchet distance. More precisely, it is a bi-criteria (α, β)-approximation algorithm in which we exceed the number of retained vertices by a factor of at most $\alpha \cdot \mathsf{OPT}$ and relax the error bound δ by a factor of β.

In Section 9.3, we have shown that there is no bi-criteria $(n^{\frac{1}{3}-\varepsilon}, 1)$-approximation algorithm (i.e., an approximation algorithm in the classical sense) for PBS for any $\varepsilon > 0$ unless $\mathsf{P} = \mathsf{NP}$. This strong inapproximability comes from the high sensitivity towards choices of keeping or discarding single vertices, which is modulated by the given value of δ. By making a bad choice, we cannot take (arbitrarily long) shortcuts that have a distance just a little greater than the given distance threshold δ to the original subpolyline. This can be overcome by relaxing the distance constraint slightly. In particular, we show that by allowing a constraint violation by a factor of $\beta = 2$, we can design an efficient algorithm with an approximation guarantee of $\alpha \in \mathcal{O}(\log(\ell + n))$. This is an exponential improvement compared to what a hypothetical classical approximation algorithm could theoretically achieve for PBS.

The key building block of our algorithm is a connection between PBS and a certain geometric set cover problem, which we call *star cover problem*. The star cover problem models the aspect of shortcutting polylines by few vertices but does not take into account consistency among different polylines. We argue, however, that approximate solutions to the star cover problem can be post-processed to form consistent PBS solutions by slightly violating the error threshold δ. For an illustration of our algorithm see Figure 9.7.

9.4.1 Star Cover Problem (StCo)

Next, we introduce the *star cover problem*, which is a special type of the set cover problem defined over instances of PBS. In the *set cover problem*, we are given a universe of objects (e.g. numbers) and a collection of subsets of this universe. The objective is to find a minimum-size set of these subsets that cover all objects in this universe. The set cover problem is known as a classical NP-complete problem.

In the star cover problem, our subsets are stars. Informally spoken, a star is a vertex together with incident (outgoing) shortcut segments of the polylines containing this vertex. To obtain a set of stars, we first direct each polyline $L \in \mathcal{L}$ in a given PBS instance $(P, \mathcal{L}, \delta)$ arbitrarily. We orient all shortcut segments of L in the same direction as L. Later we want to cover all segments of all polylines by shortcuts of stars. By directing the polylines, we can define for every vertex a unique "maximal" star and, moreover, when combining all of these stars, we can be sure that all segments are covered.

First, we define stars form formally; see Figure 9.5 for an example of a star.

Definition 9.12 (Star)**.** A *star* is the combination of a vertex $p_{\text{central}} \in P$ and, for each polyline $L \in \mathcal{L}$ that contains p_{central}, one or zero outgoing shortcut segments with respect to the distance threshold δ.

We say a star s *covers* a segment–polyline pair (e, L), if s contains a directed shortcut $\langle p_{\text{central}}, p_{\text{outer}} \rangle$ for L and e is a line segment of L coming somewhere between p_{central} and p_{outer} when traversing L. Our objective is to find a small set of stars that cover all segment–polyline pairs. We denote the set of all segment–polyline pairs in the input by $\mathcal{U}$ and the subset of segment–polyline pairs covered by a particular star s by $\mathcal{U}_s$. Then the star cover problem is defined as follows.

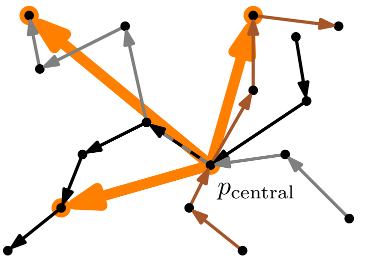


Figure 9.5: A *star* (in orange) around a vertex p_{central}, which lies on three polylines. Each polyline was assigned an arbitrary direction indicated by arrow heads.

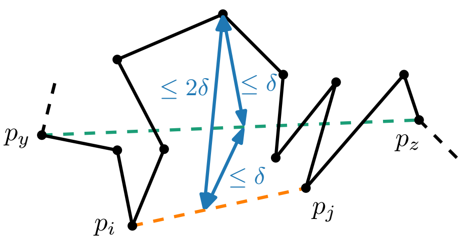


Figure 9.6: The maximum Fréchet distance between a line segment $\langle p_i, p_j \rangle$ and its corresponding subpolyline is $\leq 2\delta$ if there is a valid shortcut $\langle p_y, p_z \rangle$ going over p_i and p_j.

Definition 9.13 (Star Cover Problem (StCo))**.** A *star cover* C is a set of stars such that $\bigcup_{s \in C} \mathcal{U}_s = \mathcal{U}$, i.e., all segment–polyline pairs are covered. The *star cover problem* asks for a minimum size star cover.

Implicitly, a StCo instance depends on the distance measure we use for determining shortcuts. In this section, we only use the local Fréchet distance

9.4.2 Relationship between Solutions of PBS and StCo

Next, we investigate the relationship between an instance of PBS and a corresponding instance of StCo. (Note that to one instance of PBS, there are different instances of StCo differing only in the direction of polylines and shortcuts.) We argue that every (optimal) solution for PBS can be decomposed into a star cover. Hence, an optimal StCo solution yields a lower bound for an optimal PBS solution.

Lemma 9.14. *The size* $\mathsf{OPT}_{\text{StCo}}$ *of an optimal solution to an instance of* StCo *obtained from an instance* $(P, \mathcal{L}, \delta)$ *of* PBS *satisfies* $\mathsf{OPT}_{\text{StCo}} \leq \mathsf{OPT}_{\text{PBS}}$*, where* $\mathsf{OPT}_{\text{PBS}}$ *is the size of an optimal solution to* $(P, \mathcal{L}, \delta)$*.*

Proof. Consider an optimal solution $P^\star$ to $(P, \mathcal{L}, \delta)$. From the simplified polyline bundle induced by $P^\star$, we can get a star cover for any instance of StCo obtained from $(P, \mathcal{L}, \delta)$. First, orient all shortcuts of the simplified polyline bundle in the direction given by the StCo instance (between two vertices of $P^\star$, we may have shortcut segments in both directions on different polylines). Then, iteratively add a star in the following way. Pick a vertex $p_{\text{central}} \in P^\star$ that has at least one outgoing shortcut in the simplified polyline bundle – p_{central} becomes the central vertex of a star s. Attach to p_{central} all its outgoing shortcuts in the simplified polyline bundle to obtain the star s. Remove all shortcuts in s from the simplified polyline bundle. Repeat this procedure until there is no vertex with outgoing shortcut segments left in the simplified polyline bundle. Clearly, the obtained set of stars covers all segment-polyline pairs in $\mathcal{L}$ and is therefore a star cover. Clearly, it has at most $|P^\star|$ stars. Hence $\mathsf{OPT}_{\text{StCo}} \leq \mathsf{OPT}_{\text{PBS}}$. □

9.4.3 Approximation for StCo

We can compute an approximate solution for StCo by employing the classic greedy algorithm [Joh74] for set cover, which iteratively selects the set with the most uncovered objects until all objects are covered. However, if applied naively for StCo, the running time can be exponential in the size of the PBS instance as the number of stars can be in the order of $\Omega(n^\ell)$ (in the worst case, a star has $\Theta(n)$ choices for the endpoint of a shortcut on each of $\Theta(\ell)$ polylines). Notice, however, that it suffices to consider only *maximal stars.* A *maximal star* s uses, for each polyline L containing the central vertex of s, the outgoing shortcut segment that covers the most segments of L. As there are only n maximal stars, this guarantees polynomial running time.

Lemma 9.15. *An $\mathcal{O}(\log(t+w))$-approximation for an instance of* StCo *obtained from an instance $(P, \mathcal{L}, \delta)$ of* PBS *can be computed in $\mathcal{O}(\ell n^2 \log n)$ time, where t is the maximum number of polylines any vertex occurs in and w is the maximum number of segments any valid shortcut (according to δ) can skip.*

Proof. There is a polynomial time greedy algorithm that yields an $\mathcal{O}(\log m)$ approximation for the set cover problem, where m is the size of the largest set in the given collection of subsets of the universe [Joh74]. The greedy algorithm works as follows. While there are uncovered objects from the universe, add the set with the largest number of uncovered objects to the set cover. In an instance of StCo, m is the maximum number $\max_{\text{star } s} |\mathcal{U}_s|$ of segment–polyline pairs a single star can cover. If the central vertex of a star lies in at most t polylines, the star contains at most t shortcut segments, each of which covers at most w segments, and hence we have $m \leq tw$. Applying the greedy algorithm to this instance gives an approximation ratio in $\mathcal{O}(\log(tw)) = \mathcal{O}(\log(t+w))$.[33]

It remains to prove the polynomial running time. Using the polyline simplification algorithm for the local Fréchet distance from Chapter 8 independently for each polyline, we can find all (maximal) shortcuts for every vertex on every polyline in $\mathcal{O}(\ell n^2 \log n)$ time according to Theorem 8.12. Combining these shortcuts at every vertex gives us all n maximal stars in $\mathcal{O}(\ell n)$ time. For each star, we also store the number of segment–polyline pairs it covers and, to each segment–polyline pair, we link all stars it appears in. Both can be done in $\mathcal{O}(\ell n^2)$ time. As long as there are uncovered segment–polyline pairs, we find the star with the most uncovered segment–polyline pairs and then update the number of uncovered segment–polyline pairs for the other stars. This can be done in $\mathcal{O}(\ell n^2)$ time in total as well. □

9.4.4 Relationship between Star Covers and Solutions of PBS

While a solution for PBS can be directly converted into a star cover as argued in Section 9.4.2, the converse is more intricate. The shortcuts contained in the

[33] Note that $\mathcal{O}(\log(tw)) = \mathcal{O}(\log(t+w))$ because of the following argument. As $t, w \geq 1$, $\mathcal{O}(\log(tw)) \supseteq \mathcal{O}(\log(t+w))$ is clear. We next show that $\mathcal{O}(\log(tw)) \subseteq \mathcal{O}(\log(t+w))$. Suppose that $t \geq w$; the other case is symmetric. Then, $\mathcal{O}(\log(tw)) \subseteq \mathcal{O}(\log(t^2)) = \mathcal{O}(2\log(t)) = \mathcal{O}(\log(t)) \subseteq \mathcal{O}(\log(t+w))$.

stars of an optimal StCo solution may be overlapping or nested along a polyline, that is, vertices skipped by one shortcut may be endpoints of another shortcut in the star cover. Moreover, shared parts of multiple polylines may be covered by different stars. In other words, consistency is not guaranteed (e.g., in Figure 9.7d the shortcut of the purple star skips a vertex that is an endpoint of the other stars). We explain, however, how to derive from a star cover solution a (relaxed) solution for its corresponding instance of PBS. Some of the shortcuts of the StCo solution are replaced by "shorter" shortcuts in order to integrate some intermediate points to the PBS solution. Lemma 9.16 states that these newly introduced shortcuts can be at most 2δ away from the original polyline. The situation described there is depicted in Figure 9.6. It follows immediately from a lemma by Agarwal et al. [AHMW05].

Lemma 9.16 ([AHMW05, Lemma 3.3]). *Given a polyline $L = \langle p_1, p_2, \dots, p_{|L|}\rangle$ and a distance threshold δ. If there are $y, z \in \mathbb{N}$ with $1 \le y < z \le |L|$ such that $d_\mathrm{F}(\langle p_y, p_z\rangle, L[p_y, \dots, p_z]) \le \delta$ (i.e., segment $\langle p_y, p_z\rangle$ is a valid shortcut), then for any $i, j \in \mathbb{N}$ with $y \le i < j \le z$, $d_\mathrm{F}(\langle p_i, p_j\rangle, L[p_i, \dots, p_j]) \le 2\delta$.*

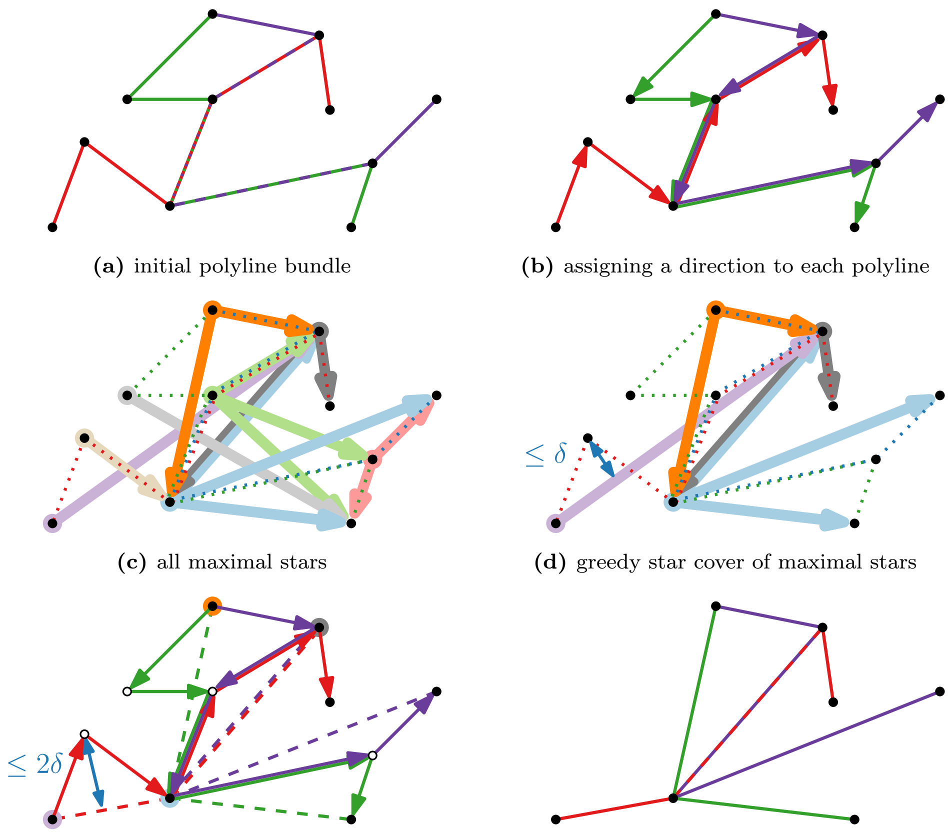
Equipped with this lemma, we now discuss the actual transformation from a StCo solution to a PBS solution with distance threshold 2δ. The idea is to keep, besides the first and last vertices of all polylines, only the central vertices of the selected stars while dropping their leaves. This is closely tied to the fact that we minimize the number of stars while ignoring their degree in the algorithm. The main insight here is that the shortcuts induced by this augmented vertex set still have a small distance to the original polylines.

Lemma 9.17. *Let C be a star cover for an instance of* StCo *obtained from an instance $(P, \mathcal{L}, \delta)$ of* PBS *under the local Fréchet distance. If C is an α-approximation for the instance of* StCo*, a bi-criteria $(\alpha + 1, 2)$-approximation for $(P, \mathcal{L}, \delta)$ under the local Fréchet distance can be computed in $\mathcal{O}(n)$ time from C.*

Proof. Let P_central be the set of central vertices of the stars in C and let P_end be the set of first and last vertices of all polylines from $\mathcal{L}$. We return $P_\mathrm{central} \cup P_\mathrm{end}$ as the bi-criteria approximate solution of $(P, \mathcal{L}, \delta)$. According to Lemma 9.14, $\mathsf{OPT}_\mathrm{StCo} \le \mathsf{OPT}_\mathrm{PBS}$. We conclude

$$|P_\mathrm{central} \cup P_\mathrm{end}| \le \alpha\mathsf{OPT}_\mathrm{StCo} + \mathsf{OPT}_\mathrm{PBS} \le (\alpha+1)\mathsf{OPT}_\mathrm{PBS}\,. \tag{9.4}$$

Let $\mathcal{L}'$ be the polyline bundle induced by $P_\mathrm{central} \cup P_\mathrm{end}$. It remains to prove that the local Fréchet distance between $\mathcal{L}'$ and $\mathcal{L}$ is at most 2δ. Consider any segment $\langle p_i, p_j\rangle$ of a (simplified) polyline $L' \in \mathcal{L}'$ corresponding to an (original) polyline $L \in \mathcal{L}$ such that p_i precedes p_j in (the directed version of) L. Notice that there is a single star s in C that covers all segments of $L[p_i, p_j]$. Otherwise, there would be another central vertex of a star between p_i and p_j on L and, in L', $\langle p_i, p_j\rangle$ would not be a segment. The central vertex p_central of s precedes p_i or is equal to p_i as otherwise s would not cover all of $L[p_i, p_j]$. Similarly, the outer vertex p_outer of s on L succeeds p_j or is equal to p_j as otherwise s would not cover all of $L[p_i, p_j]$. By the definition of a star, we know that $d_\mathrm{F}(\langle p_\mathrm{central}, p_\mathrm{outer}\rangle, L[p_\mathrm{central}, p_\mathrm{outer}]) \le \delta$. By Lemma 9.16, it follows that $d_\mathrm{F}(\langle p_i, p_j\rangle, L[p_i, p_j]) \le 2\delta$. □



(a) initial polyline bundle

(b) assigning a direction to each polyline

(c) all maximal stars

(d) greedy star cover of maximal stars

(e) retaining only vertices of $P_{\text{central}} \cup P_{\text{end}}$

(f) resulting simplified polyline bundle

Figure 9.7: Example of our bi-criteria $(\mathcal{O}(\log(\ell + n)), 2)$-approximation algorithm for PBS.

9.4.5 Bi-criteria Approximation for PBS via StCo

We have now gathered all lemmas to obtain the main theorem of this section.

Theorem 9.18. *There is a bi-criteria* $(\mathcal{O}(\log(\ell + n)), 2)$*-approximation algorithm for* PBS *under the local Fréchet distance running in* $\mathcal{O}(\ell n^2 \log n)$ *time, where* ℓ *is the number of polylines and* n *is the number of vertices in the polyline bundle.*

Proof. The steps described above provide an approximation-preserving reduction from PBS to StCo, which can be realized as a bi-criteria approximation algorithm. Its steps are depicted in Figure 9.7.

Given an instance $(P, \mathcal{L}, \delta)$ of PBS, where we let the size of an optimal solution be $\mathsf{OPT}_{\text{PBS}}$, we assign an arbitrary direction to each $L \in \mathcal{L}$ and construct the corresponding instance of StCo where we only store the maximal stars. For this corresponding instance of StCo, we compute an $\mathcal{O}(\log(t + w))$ approximation star cover C via the greedy approach described in Section 9.4.3. We can do this in

$\mathcal{O}(\ell n^2 \log n)$ time according to Lemma 9.15. According to Lemma 9.17, we can use C to compute a bi-criteria $(\mathcal{O}(\log(t+w)), 2)$-approximation for $(P, \mathcal{L}, \delta)$ in $\mathcal{O}(n)$ time. Since $t \leq \ell$ and $w \leq n$, this is also a bi-criteria $(\mathcal{O}(\log(\ell + n)), 2)$-approximation. $\square$

It is reasonable to assume that the number ℓ of polylines is polynomial in n in practically relevant settings. Hence, we essentially obtain an exponential improvement over the complexity-theoretic lower bound $n^{\frac{1}{3}-\varepsilon}$ if we allow a minor violation of the error bound. We remark that Theorem 9.18 does not carry over to the local Hausdorff distance since Lemma 9.16 does not hold for the Hausdorff distance.

9.5 Fixed-Parameter Tractability

A brute force approach to solve PBS is checking for every subset of the vertex set P in $\mathcal{O}(\ell \cdot n)$ time whether it is a valid simplification and returning the one with the fewest vertices. Consequently, the runtime of this approach is $\mathcal{O}(2^n \cdot \ell \cdot n)$. We next present a simple approach improving this runtime to FPT-time.

When considering fixed-parameter tractability (FPT), investigating parameters of the input is a natural choice. According to Corollary 9.11, PBS is not FPT in the number of polylines ℓ. However, PBS is FPT in the number of shared vertices, i.e., vertices contained in more than one polyline. We denote the set of those vertices by P_{shared} and we define $k = |P_{\text{shared}}|$.

Theorem 9.19. PBS *is fixed-parameter tractable (FPT) in the number of shared vertices k. There is an algorithm solving* PBS *in $\mathcal{O}(2^k \cdot \ell n^2)$ time under the local Hausdorff distance and in $\mathcal{O}((2^k + \log n) \cdot \ell n^2)$ time under the local Fréchet distance.*

Proof. We describe an algorithm that solves PBS in FPT-time in k. Given an instance $(P, \mathcal{L}, \delta)$ of PBS, the first step is to compute, for each $L \in \mathcal{L}$, its shortcut graph G_L using a classical polyline simplification algorithm. For the local Hausdorff distance, we can do this in $\mathcal{O}(n^2)$ time [CC96], and for the local Fréchet distance, we can do this in $\mathcal{O}(n^2 \log n)$ time according to Theorem 8.12 in Chapter 8. Hence, for all polylines of $\mathcal{L}$, we compute their shortcut graphs in time $\mathcal{O}(\ell \cdot n^2)$ and $\mathcal{O}(\ell \cdot n^2 \log n)$, respectively.

The second step is to iterate over all subsets $P' \subseteq P_{\text{shared}}$ and check if $P' = P_{\text{shared}} \cap P^\star$ where $P^\star$ is the vertex set of an optimal solution. Before the first iteration, we initialize a variable $n_{\min} = \infty$ that saves the number of vertices in the currently best solution, and we initialize a variable $\mathcal{S}_{\min} = \text{nil}$ that saves the currently best solution. Then, in each iteration, we temporarily remove from all of our shortcut graphs all nodes $P_{\text{not-contained}} = P_{\text{shared}} - P'$ and all edges that correspond to a shortcut skipping a vertex in P'. Clearly, removing $P_{\text{not-contained}}$ can be performed in $\mathcal{O}(n^2)$ time for each shortcut graph G_L. For the removal of the edges in G_L, note that we can sort the list of vertices P' and the list of all edges (defined by their endpoints) lexicographically by the occurrence of the vertices within the polyline L. If we traverse both lists simultaneously in ascending order, we remove an edge if and only if its endpoints come before and after the currently considered vertex from P'. Therefore, the removal operations can be performed in $\mathcal{O}(n^2)$ time per G_L.

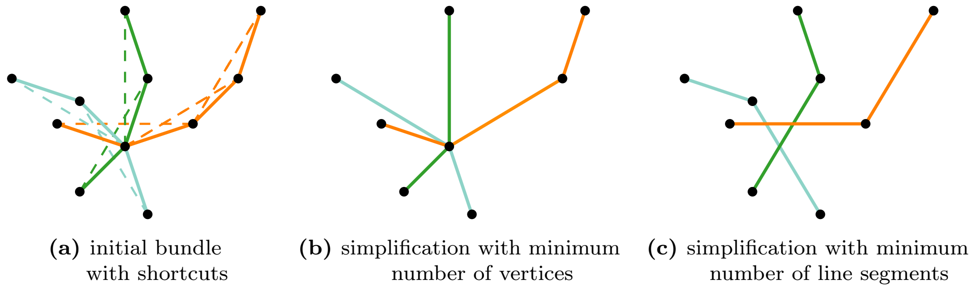

(a) initial bundle with shortcuts **(b)** simplification with minimum number of vertices **(c)** simplification with minimum number of line segments

Figure 9.8: Optimal simplifications of a polyline bundle using different objective functions.

If some shortcut graph becomes disconnected by these removal operations, we continue with the next iteration. Otherwise, we take the vertices of a shortest path from the first to the last vertex in each reduced version of G_L. This yields a simplified polyline S_L of L. Together they define a simplification $\mathcal{S}$ of our PBS instance. Observe that this simplification is consistent because each S_L contains, from the shared vertices P_{shared}, precisely the vertices in P'.

If the number $n_{\mathcal{S}}$ of vertices in $\mathcal{S}$ is less than $n_{\min}$, we set $n_{\min} = n_{\mathcal{S}}$ and $\mathcal{S}_{\min} = \mathcal{S}$. In the end, we return $\mathcal{S}_{\min}$. Since we have 2^k subsets of P_{shared} and each iteration can be performed in $\mathcal{O}(\ell \cdot n^2)$ time, the running time of the second step is in $\mathcal{O}(2^k \cdot \ell \cdot n^2)$.

First, note that our algorithm always returns some polyline simplification because for $P' = P_{\text{shared}}$, we do not get a disconnected G_L after the removal operations. It remains to prove that $\mathcal{S}_{\min}$ is an optimal solution of our input instance of PBS. The returned solution is valid because the shared vertices of P' are retained in all simplified polylines (they cannot be skipped) and the other shared vertices are skipped in all simplified polylines. Our algorithm finds the minimum size solution because in one iteration it considers $P' = P_{\text{shared}} \cap P^\star$. Moreover, an optimal solution cannot have fewer vertices occurring in only one polyline L than our algorithm since this would imply a shorter shortest path within the reduced version of G_L. □

9.6 Concluding Remarks and Open Problems

We have generalized the well-known problem of polyline simplification from a single polyline to multiple polylines sharing vertices and line segments, which we have called a polyline bundle. Although efficient algorithms for a single polyline have been known for a long time (we have also seen one in the previous chapter), we could show that simplifying two or more polylines consistently is a problem that is NP-hard to approximate within a factor of $n^{\frac{1}{3}-\varepsilon}$ for any $\varepsilon > 0$ where n is the number of vertices in the polyline bundle.

However, if we relax the constraint on the Fréchet distance between original and simplified polyline by a factor of 2, we can overcome this strong inapproximability bound. We remark that Bosch et al. [BSS+21] have implemented the bi-criteria approximation algorithm from Section 9.4 to do an experimental evaluation. They have found out that the algorithm indeed exceeds the distance threshold bound δ by a factor of 2 on practically-relevant instances but still produces high-quality solutions. This suggests the applicability of this (originally only theoretical) algorithm on real-world instances. Moreover, we have seen that we can find an optimal simplification efficiently if we have only a small number of shared vertices since the problem of polyline bundle simplification is fixed-parameter tractable (FPT) in this parameter.

Based on our results, there are many possible directions for future research. Specifically, we propose the following open problems.

- Improve our inapproximability bound of $n^{\frac{1}{3}-\varepsilon}$ further or show its tightness. Also investigate the landscape of possible and impossible bi-criteria (α, β)-approximation algorithms.
- Our current bi-criteria approximation guarantee is logarithmic in the number of polylines ℓ plus the number of vertices n. In most practical applications, ℓ is smaller than n or at most polynomial in n. From a theoretical perspective, however, it might be interesting to get rid of the dependency on ℓ in the bi-criteria approximation in order to get improvements for the case where ℓ is significantly larger than n.
- As a distance measure, we have employed the local Fréchet distance, which we consider to be more natural and intuitive than the local Hausdorff distance when comparing polylines. However, the local Hausdorff distance is sometimes used in classical polyline simplification as well. Our hardness result and the FPT-algorithm also apply to the Hausdorff distance, but our bi-criteria approximation algorithm fails since Lemma 9.16 is not applicable for the Hausdorff distance. Consider PBS using the local Hausdorff distance or other (even non-local) distance measures.
- In our generalization from a single polyline to a polyline bundle, the objective is to minimize the number of retained vertices. However, minimizing the number of retained line segments is an alternative objective function, which also generalizes the classical minimization problem for a single polyline. Optimal simplifications for both objectives may differ; see Figure 9.8. Our hardness and our findings regarding FPT-algorithms also hold when minimizing the number of retained line segments. However, it is not clear how to obtain a similar result for the bi-criteria approximability.
- A severe restriction in our definition for simplifying polyline bundles is that the endpoints of each individual polyline need to be kept. For many instances, this reduces the number of possible simplifications significantly, which makes investigating a model that allows removing endpoints of polylines even more important for polyline bundles than for a single polyline, where we have also suggested this as an open problem (see Section 8.6).
- Do experimental evaluations (incorporating the work by Bosch et al. [BSS+21]) on real-world data (also in other domains) and develop new heuristic, approximation, or exact algorithms for comparison.

Chapter 10
Conclusion

In the previous seven chapters, we have presented seven problems from the field of graph drawing and polyline simplification. A recurring theme across the chapters was the use of simple geometric objects based on line segments. We have seen that geometric problems with such simple structure can often be reduced to a discrete problem, usually a combinatorial or a graph problem, which makes them more elegant to describe, more pleasant to work with, and easier to classify.

Also in the places where we had both line segments and circular arcs or squares and circles, working with line segments or squares was easier and gave us tighter bounds. In Chapter 4, we could only show a general arc-based bound for the limited class of maximal outerpaths, and in Chapter 8 handling circle-shaped unit disks in the L_2-norm was more difficult and increased the runtime of the algorithm in comparison to the L_1- and L_∞-norms where we have square-shaped unit disks.

Often enough, the problems we have worked on, though of seemingly simple appearance, turned out to be algorithmically hard to solve. In this book, we have presented five different hardness proofs. In contrast to positive results, their practical utility is limited and there are usually many completely different constructions with the same result possible. However, having a nice and pleasant construction can also be an artful thing. The cover of this book shows one of its hardness constructions. Moreover, from a more rational viewpoint, knowing whether a problem is NP-hard or hard to approximate is a necessary information to know what algorithm we can seek for.

We have also seen some applied parts that originated from an exchange with industrial partners where both sides could benefit from: our industrial partners had the demand of programs drawing good cable and network plans automatically. It turned out that some theory-based well-studied approaches in graph drawing could, at least as a first solution, afford most of it. For the remaining parts, this gave us new interesting theoretical problems: Chapter 6, where we consider proper colorings of mixed interval graphs, would not exist if we had not come across this problem by working on the layer-based drawing algorithm from Chapter 7, with which we have drawn cable plans. When working on a theoretical problem, it is also a good thing to have an application for it or related to it in mind as this can prevent a somewhat too strong decoupling of theory and practice.

Finally, we want to summarize our findings and recount our most important open problems.

We have started this book with stick graphs (see Chapter 3), which is a problem where many open questions have now been settled in the last four years – either by us or by other authors. Our main result is that the problem of recognizing a stick graph with given stick lengths is NP-complete, even if we have just three different stick length or if the internal order of the horizontal and the vertical sticks is given. Quite interestingly, the latter is only true if we have isolated vertices. So, already with many small connected components (each of size at least 2) the problem becomes polynomial-time solvable, which is quite an interesting boundary to occur. We have also introduced a data structure called *semi-ordered tree*, which might be interesting to apply in a different context.

We have continued investigating the segment number (see Chapter 4), where little attention has been given to universal lower bounds for planar graph classes so far. We could now show that the number of segments required in any planar straight-line drawing of any n-vertex maximal outerpath, 2-tree, and planar 3-tree is at least $n/2 + 2$, $(n + 7)/5$, and $n + 4$, respectively. For a cactus graph G, we can describe a linear-time algorithm drawing G with the minimum number of segments, which generalizes a result for trees. We remark that maximal outerplanar graphs are also 2-trees. However, we have not found an example close to our lower bound of 2-trees because it is most likely not tight.

Open Problem 10.1. Find a tight universal lower bound on the segment number of maximal outerplanar graphs or, more general, 2-trees.

Also, we were initially motivated by the question of how much we can save if we use circular arcs instead of line segments. We have found infinitely many planar graphs where the segment number of a graph is three times greater than its arc number, but for no graph we have found a greater ratio.

Open Problem 10.2. Show that the segment number of a graph is at most three times its arc number or show the contrary – ideally by giving an explicit counterexample.

When studying the segment number of outerpaths, we also investigated the arc number and the more general concept of pseudo-k-arc arrangements where each pair of curves intersects at most k times. However, we did not delve very deep into the subject matter.

Open Problem 10.3. Study arrangements of pseudo-k-arcs.

Besides the segment number, the slope number is a measure for the visual complexity of a graph drawing. We have studied the problem of finding a straight-line upward-planar drawing of a given directed graph if we are allowed to use at most k slopes (see Chapter 5). Fortunately, we could settle the decision version of this question for some important graph classes. We found that for $k = 3$, it is polynomial-time solvable for cactus digraphs and directed inner triangulations, but NP-hard to decide for directed outerpaths even if its faces are only triangles and quadrilaterals. For $k > 3$, the gap widens to cactus digraphs on the efficient side and planar digraphs on the NP-hard side.

Open Problem 10.4. For $k > 3$, is it NP-hard to decide whether a directed outerpath or a directed outerplanar graph admits an upward-planar straight-line drawing with k slopes?

Also, the connection of the segment number and upward-planar straight-line drawings could be interesting.

Open Problem 10.5. Study the *upward-planar segment number.*

Motivated by the application of routing edges orthogonally between layers on few horizontal tracks, we have investigated the problem of coloring mixed interval graphs (see Chapter 6). In the application, we need to color bidirectional interval graphs. However, we have only shown that for a (simpler) n-vertex directional interval graph, we can find an optimal proper coloring in $\mathcal{O}(n \log n)$ time if an interval representation is given (and in $\mathcal{O}(n^2)$ time otherwise). This yields a simple 2-approximation for bidirectional interval graphs. However, it is not clear whether we can also find an optimal proper coloring for bidirectional interval graphs efficiently.

Open Problem 10.6. Can we find a proper coloring with the minimum number of colors for a bidirectional interval graph efficiently or is this problem NP-complete?

Even if it is NP-complete, there seems to be hope that we can do better than a 2-approximation.

Open Problem 10.7. Find an α-approximation algorithm with $\alpha < 2$ for proper colorings of bidirectional interval graphs or show that such an algorithm cannot exist.

Also an alternative definition of directional interval graphs might be an interesting path of research to follow.

Open Problem 10.8. Study mixed interval graphs where we have a directed edge for a contained interval and an undirected edge otherwise.

Building upon the Sugiyama-framework, we have seen that we can automatically compute an orthogonal port-based layout of a technical diagram (see Chapter 7) with port pairings and port groups, which may represent, for instance, plugs and plug sockets. The resulting drawing fulfills specific aesthetic expectations and compares well with an established library of similar drawing capability. However, there is still room for improvement.

Open Problem 10.9. Extend and improve our Sugiyama-based algorithm for drawing undirected graphs with generalized port constraints.

In the realm of polylines, we have seen that simplifying an n-vertex polyline optimally under the local Fréchet distances can be done in $\mathcal{O}(n^2 \log n)$ time (see Chapter 8). An algorithm that retains only the minimum number of vertices in a simplified polyline is also known as an *Imai–Iri*(-based) algorithm. It is however not clear if this is now an asymptotically optimal algorithm.

Open Problem 10.10. Find an algorithm for simplifying an n-vertex polyline under the local Fréchet distance (in the L_2-norm) retaining the minimum number of vertices that runs in $o(n^2 \log n)$ time or show that $\mathcal{O}(n^2 \log n)$ is worst-case optimal.

From a theoretical perspective, generalizing this result to higher dimensions can also be interesting.

Open Problem 10.11. Find an algorithm for simplifying an n-vertex polyline in higher dimensions (e.g., start with three dimensions) under the local Fréchet distance retaining the minimum number of vertices that runs in $o(n^3)$ time or show that this is not possible.

We have generalized single polylines to bundles of polylines (see Chapter 9) and argued why we require a *consistent* simplification.

Open Problem 10.12. Find more applications to use polyline bundle simplification in practice.

In contrast to single polylines, the problem becomes even hard to approximate. Namely, simplifying an n-vertex polyline bundle under the local Hausdorff or Fréchet distance retaining the minimum number of vertices is NP-hard to approximate within a factor of $n^{\frac{1}{3}-\varepsilon}$ for any $\varepsilon > 0$. We have seen that this holds true even if the input polyline bundle is planar and consists of only two polylines. To overcome this hardness, we have introduced a bi-criteria approximation algorithm. We could show that simplifying an n-vertex polyline bundle of ℓ polylines under the local Fréchet distance retaining the minimum number of vertices can be approximated within a factor of $\mathcal{O}(\log(\ell + n))$ if we allow the distance threshold to be violated by a factor of 2.

Open Problem 10.13. Investigate the combinations of α and β for which there are (no) bi-criteria (α, β)-approximation algorithms for polyline bundle simplification.

Bibliography

[Adn08] Muhammad Abdullah Adnan. Minimum segment drawings of outerplanar graphs. Master's thesis, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology (BUET), Dhaka, 2008. (accessed on 2022/12/08). URL: http://lib.buet.ac.bd:8080/xmlui/bitstream/handle/123456789/1565/Full%20%20Thesis%20.pdf. 59, 60

[AG95] Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry and Applications*, 5:75–91, 1995. doi:10.1142/S0218195995000064. 190

[AHMW05] Pankaj K. Agarwal, Sariel Har-Peled, Nabil H. Mustafa, and Yusu Wang. Near-linear time approximation algorithms for curve simplification. *Algorithmica*, 42(3–4):203–219, 2005. doi:10.1007/s00453-005-1165-y. 186, 188, 213, 230

[AKPW15] Mahmuda Ahmed, Sophia Karagiorgou, Dieter Pfoser, and Carola Wenk. *Map Construction Algorithms.* Springer, 2015. doi:10.1007/978-3-319-25166-0. 185, 213

[AV00] Pankaj K. Agarwal and Kasturi R. Varadarajan. Efficient algorithms for approximating polygonal chains. *Discrete & Computational Geometry*, 23(2):273–291, 2000. doi:10.1007/PL00009500. 187

[AW12] Mahmuda Ahmed and Carola Wenk. Constructing street networks from GPS trajectories. In *Proc. 20th Annual European Symposium on Algorithms (ESA'12)*, pages 60–71. Springer, 2012. doi:10.1007/978-3-642-33090-2_7. 185

[BBB$^+$08] Christian Bachmaier, Franz J. Brandenburg, Wolfgang Brunner, Andreas Hofmeier, Marco Matzeder, and Thomas Unfried. Tree drawings on the hexagonal grid. In *Proc. 16th International Symposium on Graph Drawing (GD'08)*, pages 372–383. Springer, 2008. doi:10.1007/978-3-642-00219-9_36. 92, 95

[BBK$^+$16] Kevin Buchin, Maike Buchin, Maximilian Konzack, Wolfgang Mulzer, and André Schulz. Fine-grained analysis of problems on curves. In *Proc. 32nd European Workshop on Computational Geometry (EuroCG'16)*, 2016. URL: https://www.eurocg2016.usi.ch/sites/default/files/paper_68.pdf. 187

[BBK$^+$20] Milutin Brankovic, Kevin Buchin, Koen Klaren, André Nusser, Aleksandr Popov, and Sampson Wong. (k, l)-medians clustering of trajectories using continuous dynamic time warping. In *Proc. 28th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 99–110, 2020. `doi:10.1145/3397536.3422245`. 185, 209

[BC21] Karl Bringmann and Bhaskar Ray Chaudhury. Polyline simplification has cubic complexity. *Journal of Computational Geometry*, 11(2):94–130, 2021. `doi:10.20382/jocg.v11i2a5`. 186, 187

[BCD$^+$02] Gill Barequet, Danny Z. Chen, Ovidiu Daescu, Michael T. Goodrich, and Jack Snoeyink. Efficiently approximating polygonal paths in three and higher dimensions. *Algorithmica*, 33(2):150–167, 2002. `doi:10.1007/s00453-001-0096-5`. 187

[BD16] Carla Binucci and Walter Didimo. Computing quasi-upward planar drawings of mixed graphs. *The Computer Journal*, 59(1):133–150, 2016. `doi:10.1093/comjnl/bxv082`. 126

[BDD$^+$18] Michael A. Bekos, Emilio Di Giacomo, Walter Didimo, Giuseppe Liotta, and Fabrizio Montecchiani. Universal slope sets for upward planar drawings. In *Proc. 26th International Symposium on Graph Drawing & Network Visualization (GD'18)*, pages 77–91, 2018. `doi:10.1007/978-3-030-04414-5_6`. 91

[BDG$^+$19] Kevin Buchin, Anne Driemel, Joachim Gudmundsson, Michael Horton, Irina Kostitsyna, Maarten Löffler, and Martijn Struijs. Approximating (*k, l*)-center clustering for curves. In *Proc. 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'19)*, pages 2922–2938. SIAM, 2019. `doi:10.1137/1.9781611975482.181`. 214

[BDH$^+$04] Ron Breukelaar, Erik D. Demaine, Susan Hohenberger, Hendrik Jan Hoogeboom, Walter A. Kosters, and David Liben-Nowell. Tetris is hard, even to approximate. *International Journal of Computational Geometry and Applications*, 14(1–2):41–68, 2004. `doi:10.1142/S0218195904001354`. 2

[BDLM94] Paola Bertolazzi, Giuseppe Di Battista, Giuseppe Liotta, and Carlo Mannino. Upward drawings of triconnected digraphs. *Algorithmica*, 12(6):476–497, 1994. `doi:10.1007/BF01188716`. 24, 91

[BDMT98] Paola Bertolazzi, Giuseppe Di Battista, Carlo Mannino, and Roberto Tamassia. Optimal upward planarity testing of single-source digraphs. *SIAM Journal on Computing*, 27(1):132–169, 1998. `doi:10.1137/S0097539794279626`. 91

[BDP14] Carla Binucci, Walter Didimo, and Maurizio Patrignani. Upward and quasi-upward planarity testing of embedded mixed graphs. *Theoretical Computer Science*, 526:75–89, 2014. `doi:10.1016/j.tcs.2014.01.015`. 126

[BDR23] Maike Buchin, Anne Driemel, and Dennis Rohde. Approximating (k, l)-median clustering for polygonal curves. *ACM Trans. Algorithms*, 19(1):4:1–4:32, 2023. `doi:10.1145/3559764`. 214

[BDS20] Kevin Buchin, Anne Driemel, and Martijn Struijs. On the hardness of computing an average curve. In *Proc. 17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT'20)*, pages 19:1–19:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.SWAT.2020.19`. 214

[BFM06] Michael A. Bender, Martin Farach-Colton, and Miguel A. Mosteiro. Insertion sort is $O(n \log n)$. *Theory of Computing Systems*, 39(3):391–397, 2006. `doi:10.1007/s00224-005-1237-z`. 1

[BHW$^+$21] Michael Burch, Weidong Huang, Mathew Wakefield, Helen C. Purchase, Daniel Weiskopf, and Jie Hua. The state of the art in empirical user evaluation of graph visualizations. *IEEE Access*, 9:4173–4198, 2021. `doi:10.1109/ACCESS.2020.3047616`. 3

[BHZ18] Jørgen Bang-Jensen, Jing Huang, and Xuding Zhu. Completing orientations of partially oriented graphs. *Journal of Graph Theory*, 87(3):285–304, 2018. `doi:10.1002/jgt.22157`. 126

[BJW$^+$08] Sergey Bereg, Minghui Jiang, Wencheng Wang, Boting Yang, and Binhai Zhu. Simplifying 3D polygonal chains under the discrete Fréchet distance. In *Proc. 8th Latin American Symposium on Theoretical Informatics (LATIN'08)*, pages 630–641. Springer, 2008. `doi:10.1007/978-3-540-78773-0_54`. 188, 212

[BK02] Ulrik Brandes and Boris Köpf. Fast and simple horizontal coordinate assignment. In *Proc. 9th International Symposium on Graph Drawing (GD'01)*, pages 31–44. Springer, 2002. `doi:10.1007/3-540-45848-4\_3`. 145, 151, 152, 162, 167

[BKK18] Maike Buchin, Bernhard Kilgus, and Andrea Kölzsch. Group diagrams for representing trajectories. In *Proc. 11th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, pages 1–10, 2018. `doi:10.1145/3283207.3283208`. 213

[BKM19] Guido Brückner, Nadine Davina Krisam, and Tamara Mchedlidze. Level-planar drawings with few slopes. In *Proc. 27th International Symposium on Graph Drawing & Network Visualization (GD'19)*, pages 559–572. Springer, 2019. `doi:10.1007/978-3-030-35802-0_42`. 91

[BL76] Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976. doi:10.1016/S0022-0000(76)80045-1. 37

[BLS99] Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes: A Survey.* Discrete Mathematics and Applications. SIAM, 1999. doi:10.1137/1.9780898719796. 31

[BM10] Wolfgang Brunner and Marco Matzeder. Drawing ordered $(k-1)$-ary trees on k-grids. In *Proc. 18th International Symposium on Graph Drawing (GD'10)*, pages 105–116. Springer, 2010. doi:10.1007/978-3-642-18469-7_10. 92, 95

[BM13] Christian Bachmaier and Marco Matzeder. Drawing unordered trees on k-grids. *Journal of Graph Algorithms and Applications*, 17(2):103–128, 2013. doi:10.7155/jgaa.00287. 92, 95

[BMT00] Therese C. Biedl, Brendan Madden, and Ioannis G. Tollis. The three-phase method: A unified approach to orthogonal graph drawing. *International Journal of Computational Geometry and Applications*, 10(6):553–580, 2000. doi:10.1142/S0218195900000310. 143

[Brü21] Lukas Brückner. Orthogonal drawing as a coloring problem in perfect graphs. Bachelor's thesis, Institut für Informatik, Universität Würzburg, 2021. (in German). URL: https://www1.pub.informatik.uni-wuerzburg.de/pub/theses/2021-brueckner-bachelor.pdf. 132

[BSS$^+$21] Yannick Bosch, Peter Schäfer, Joachim Spoerhase, Sabine Storandt, and Johannes Zink. Consistent simplification of polyline tree bundles. In *Proc. 27th International Computing and Combinatorics Conference (COCOON'21)*, pages 231–243. Springer, 2021. doi:10.1007/978-3-030-89543-3_20. 5, 13, 209, 233, 234

[BvdHO$^+$22] Maike Buchin, Ivor van der Hoog, Tim Ophelders, Lena Schlipf, Rodrigo I. Silveira, and Frank Staals. Efficient Fréchet distance queries for segments. In *Proc. 30th Annual European Symposium on Algorithms (ESA'22)*, pages 29:1–29:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ESA.2022.29. 11, 186, 187, 189, 208

[BvdSZ$^+$02] R. W. Bulterman, F. W. van der Sommen, Gerard Zwaan, Tom Verhoeff, Antonetta J. M. van Gasteren, and Wim H. J. Feijen. On computing a longest path in a tree. *Information Processing Letters*, 81(2):93–96, 2002. doi:10.1016/S0020-0190(01)00198-3. 97

[BWZ20] Ulrik Brandes, Julian Walter, and Johannes Zink. Erratum: Fast and simple horizontal coordinate assignment. arXiv preprint, 2020. `arXiv:2008.01252`. 145, 152

[CC96] W. S. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments or minimum error. *International Journal of Computational Geometry and Applications*, 6(1):59–77, 1996. `doi:10.1142/S0218195996000058`. 11, 186, 187, 190, 194, 198, 232

[CCF$^+$17] Daniele Catanzaro, Steven Chaplick, Stefan Felsner, Bjarni V. Halldórsson, Magnús M. Halldórsson, Thomas Hixon, and Juraj Stacho. Max point-tolerance graphs. *Discrete Applied Mathematics*, 216:84–97, 2017. `doi:10.1016/j.dam.2015.08.019`. 33

[CDF$^+$22] Steven Chaplick, Emilio Di Giacomo, Fabrizio Frati, Robert Ganian, Chrysanthi N. Raftopoulou, and Kirill Simonov. Parameterized algorithms for upward planarity. In *Proc. 38th International Symposium on Computational Geometry (SoCG'22)*, pages 26:1–26:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. `doi:10.4230/LIPIcs.SoCG.2022.26`. 91

[CDK$^+$14] Steven Chaplick, Paul Dorbec, Jan Kratochvíl, Mickaël Montassier, and Juraj Stacho. Contact representations of planar graphs: Extending a partial representation is hard. In *Proc. 40th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'14)*, pages 139–151. Springer, 2014. `doi:10.1007/978-3-319-12340-0\_12`. 33

[CDP92] Pierluigi Crescenzi, Giuseppe Di Battista, and Adolfo Piperno. A note on optimal area algorithms for upward drawings of binary trees. *Computational Geometry*, 2(4):187–200, 1992. `doi:10.1016/0925-7721(92)90021-J`. 92

[CFHW18] Steven Chaplick, Stefan Felsner, Udo Hoffmann, and Veit Wiechert. Grid intersection graphs and order dimension. *Order*, 35(2):363–391, 2018. `doi:10.1007/s11083-017-9437-0`. 6, 33

[CFKW20] Steven Chaplick, Henry Förster, Myroslav Kryven, and Alexander Wolff. Drawing graphs with circular arcs and right-angle crossings. In *Proc. 17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT'20)*, pages 21:1–21:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.SWAT.2020.21`. 60

[CFL$^+$17] Steven Chaplick, Krzysztof Fleszar, Fabian Lipp, Alexander Ravsky, Oleg Verbitsky, and Alexander Wolff. The complexity of drawing graphs on few lines and few planes. In *Proc. 15th Algorithms and Data Structures Symposium (WADS'17)*, pages 265–276. Springer, 2017. `doi:10.1007/978-3-319-62127-2_23`. 60

[CFL19] Henning Cordes, Bryan Foltice, and Thomas Langer. Misperception of exponential growth: Are people aware of their errors? *Decision Analysis*, 16(4):261–280, 2019. `doi:10.1287/deca.2019.0395`. 4

[CFL$^+$20] Steven Chaplick, Krzysztof Fleszar, Fabian Lipp, Alexander Ravsky, Oleg Verbitsky, and Alexander Wolff. Drawing graphs on few lines and few planes. *Journal of Computational Geometry*, 11(1):433–475, 2020. `doi:10.20382/jocg.v11i1a17`. 60

[CFM$^+$18] Jean Cardinal, Stefan Felsner, Tillmann Miltzow, Casey Tompkins, and Birgit Vogtenhuber. Intersection graphs of rays and grounded segments. *Journal of Graph Algorithms and Applications*, 22(2):273–295, 2018. `doi:10.7155/jgaa.00470`. 33

[CG09] Jérémie Chalopin and Daniel Gonçalves. Every planar graph is the intersection graph of segments in the plane: Extended abstract. In *Proc. 41st ACM Symposium on Theory of Computing (STOC'09)*, pages 631–638. ACM, 2009. `doi:10.1145/1536414.1536500`. 32

[CGM$^+$10] Markus Chimani, Carsten Gutwenger, Petra Mutzel, Miro Spönemann, and Hoi-Ming Wong. Crossing minimization and layouts of directed hypergraphs with port constraints. In *Proc. 18th International Symposium on Graph Drawing (GD'10)*, pages 141–152. Springer, 2010. `doi:10.1007/978-3-642-18469-7_13`. 143

[CGMW10] Markus Chimani, Carsten Gutwenger, Petra Mutzel, and Hoi-Ming Wong. Layer-free upward crossing minimization. *ACM J. Exp. Algorithmics*, 15(2.2):2.1–2.27, 2010. `doi:10.1145/1671970.1671975`. 143

[CH23] Siu-Wing Cheng and Haoqiang Huang. Curve simplification and clustering under Fréchet distance. In *Proc. 34th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'23)*, pages 1414–1432. SIAM, 2023. `doi:10.1137/1.9781611977554.ch51`. 214

[Cha04] Hubert Chan. A parameterized algorithm for upward planarity testing. In *Proc. 12th Annual European Symposium on Algorithms (ESA'04)*, pages 157–168, 2004. `doi:10.1007/978-3-540-30140-0_16`. 91

[CHO$^+$17] Steven Chaplick, Pavol Hell, Yota Otachi, Toshiki Saitoh, and Ryuhei Uehara. Ferrers dimension of grid intersection graphs. *Discrete Applied Mathematics*, 216:130–135, 2017. `doi:10.1016/j.dam.2015.05.035`. 33

[CJ17] Sergio Cabello and Miha Jejčič. Refining the hierarchies of classes of geometric intersection graphs. *Electronic Journal of Combinatorics*, 24(1):P1.33, 2017. `doi:10.37236/6040`. 6, 33, 35

[CKL$^+$20] Steven Chaplick, Philipp Kindermann, Andre Löffler, Florian Thiele, Alexander Wolff, Alexander Zaft, and Johannes Zink. Recognizing stick graphs with and without length constraints. *Journal of Graph Algorithms and Applications*, 24(4):657–681, 2020. `doi:10.7155/jgaa.00524`. 6, 35

[CLRS22] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 4th edition, 2022. 15, 54

[CPR90] Jurek Czyzowicz, Andrzej Pelc, and Ivan Rival. Drawing orders with few slopes. *Discrete Mathematics*, 82(3):233–250, 1990. `doi:10.1016/0012-365X(90)90201-R`. 92

[CR85] Joseph C. Culberson and Gregory J. E. Rawlins. Turtlegons: generating simple polygons for sequences of angles. In *Proc. 1st Symposium on Computational Geometry (SoCG'85)*, pages 305–310. ACM, 1985. `doi:10.1145/323233.323272`. 92, 102, 103, 104

[Čul64] Karel Čulík. Applications of graph theory to mathematical logic and linguistics. In *Theory of Graphs and its Applications (Proceedings of the Symposium held in Smolenice 1963)*, 1964. 31

[Czy91] Jurek Czyzowicz. Lattice diagrams with few slopes. *Journal of Combinatorial Theory, Series A*, 56(1):96–108, 1991. `doi:10.1016/0097-3165(91)90025-C`. 92

[Dam19] Peter Damaschke. Parameterized mixed graph coloring. *Journal of Combinatorial Optimization*, 38:362–374, 2019. `doi:10.1007/s10878-019-00388-z`. 126

[dBK12] Mark de Berg and Amirali Khosravi. Optimal binary space partitions for segments in the plane. *International Journal of Computational Geometry and Applications*, 22(3):187–206, 2012. `doi:10.1142/s0218195912500045`. 109

[dBvKS98] Marc de Berg, Marc J. van Kreveld, and Stefan Schirra. Topologically correct subdivision simplification using the bandwidth criterion. *Cartography and Geographic Information Systems*, 25(1):243–257, 1998. `doi:10.1559/152304098782383007`. 213

[DESW07] Vida Dujmović, David Eppstein, Matthew Suderman, and David R. Wood. Drawings of planar graphs with few slopes and segments. *Computational Geometry*, 38(3):194–212, 2007. `doi:10.1016/j.comgeo.2006.09.002`. 7, 57, 58, 60, 81, 82, 85, 91, 95

[DETT99] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999. 57, 90, 94 [dFPP90] Hubert de Fraysseix, János Pach, and Richard Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990. `doi:10.1007/bf02122694`. 24

[DGL10] Walter Didimo, Francesco Giordano, and Giuseppe Liotta. Upward spirality and upward planarity testing. *SIAM Journal on Discrete Mathematics*, 23(4):1842–1899, 2010. `doi:10.1137/070696854`. 91

[DH73] Richard O. Duda and Peter E. Hart. *Pattern classification and scene analysis.* A Wiley-Interscience publication. Wiley, 1973. 185

[DHK$^+$19] Felice De Luca, Md. Iqbal Hossain, Stephen G. Kobourov, Anna Lubiw, and Debajyoti Mondal. Recognition and drawing of stick graphs. *Theoretical Computer Science*, 796:22–33, 2019. `doi:10.1016/j.tcs.2019.08.018`. 6, 33, 35, 36

[DHW12] Anne Driemel, Sariel Har-Peled, and Carola Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete & Computational Geometry*, 48(1):94–127, 2012. `doi:10.1145/1810959.1811019`. 208

[DKS16] Anne Driemel, Amer Krivošija, and Christian Sohler. Clustering time series under the Fréchet distance. In *Proc. 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'16)*, pages 766–785. SIAM, 2016. `doi:10.1137/1.9781611974331.ch55`. 213

[DLM15] Emilio Di Giacomo, Giuseppe Liotta, and Fabrizio Montecchiani. Drawing outer 1-planar graphs with few slopes. *Journal of Graph Algorithms and Applications*, 19(2):707–741, 2015. `doi:10.7155/jgaa.00376`. 91

[DLM18] Emilio Di Giacomo, Giuseppe Liotta, and Fabrizio Montecchiani. Drawing subcubic planar graphs with four slopes and optimal angular resolution. *Theoretical Computer Science*, 714:51–73, 2018. `doi:10.1016/j.tcs.2017.12.004`. 91, 94

[DLM20] Emilio Di Giacomo, Giuseppe Liotta, and Fabrizio Montecchiani. 1-bend upward planar slope number of SP-digraphs. *Computational Geometry*, 90:101628, 2020. `doi:10.1016/j.comgeo.2020.101628`. 91

[DM19] Stephane Durocher and Debajyoti Mondal. Drawing plane triangulations with few segments. *Computational Geometry*, 77:27–39, 2019. `doi:10.1016/j.comgeo.2018.02.003`. 58, 59

[DMNW13] Stephane Durocher, Debajyoti Mondal, Rahnuma Islam Nishat, and Sue Whitesides. A note on minimum-segment drawings of planar graphs. *Journal of Graph Algorithms and Applications*, 17:301–328, 2013. `doi:10.7155/jgaa.00295`. 59

[DP73] David H. Douglas and Thomas K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica*, 10(2):112–122, 1973. `doi:10.1002/9780470669488.ch2`. 188

[DSW07] Vida Dujmović, Matthew Suderman, and David R. Wood. Graph drawings with few slopes. *Computational Geometry*, 38(3):181–193, 2007. `doi:10.1016/j.comgeo.2006.08.002`. 91

[DT88] Giuseppe Di Battista and Roberto Tamassia. Algorithms for plane representations of acyclic digraphs. *Theoretical Computer Science*, 61(2):175–198, 1988. `doi:10.1016/0304-3975(88)90123-5`. 24, 91

[EET76] Gideon Ehrlich, Shimon Even, and Robert Endre Tarjan. Intersection graphs of curves in the plane. *Journal of Combinatorial Theory, Series B*, 21(1):8–20, 1976. `doi:10.1016/0095-8956(76)90022-8`. 31

[EGP66] Paul Erdős, A. W. Goodman, and Louis Pósa. The representation of a graph by set intersections. *Canadian Journal of Mathematics*, 18:106–112, 1966. `doi:10.4153/CJM-1966-014-3`. 31

[elk20] Eclipse layout kernel (ELK), 2020. (accessed on September 21, 2022). URL: `https://www.eclipse.org/elk/`. 143, 162

[EM01] Regina Estkowski and Joseph S. B. Mitchell. Simplifying a polygonal subdivision while keeping it simple. In *Proc. 17th Annual Symposium on Computational Geometry (SoCG'01)*, pages 40–49. ACM, 2001. `doi:10.1145/378583.378612`. 213, 215

[EW94] Peter Eades and Sue Whitesides. Drawing graphs in two layers. *Theoretical Computer Science*, 131(2):361–374, 1994. `doi:10.1016/0304-3975(94)90179-1`. 145

[Fár48] István Fáry. On straight-line representation of planar graphs. *Acta Scientiarum Mathematicarum (Szeged)*, 11:229–233, 1948. 24

[FFK$^+$15] Chenglin Fan, Omrit Filtser, Matthew J. Katz, Tim Wylie, and Binhai Zhu. On the chain pair simplification problem. In *Proc. 14th International Symposium on Algorithms and Data Structures (WADS'15)*, pages 351–362. Springer, 2015. `doi:10.1007/978-3-319-21840-3_29`. 212

[FFKZ16] Chenglin Fan, Omrit Filtser, Matthew J. Katz, and Binhai Zhu. On the general chain pair simplification problem. In *Proc. 41st International Symposium on Mathematical Foundations of Computer Science (MFCS'16)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. `doi:10.4230/LIPIcs.MFCS.2016.37`. 212

[FG17] Stefan Felsner and Jacob E. Goodman. Pseudoline arrangements. In Jacob E Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 5, pages 125–158. CRC Press LLC, 2017. 61

[FGR04] Patrick C. Fricker, Marcus Gastreich, and Matthias Rarey. Automated drawing of structural molecular formulas under constraints. *Journal of Chemical Information and Modeling*, 44(3):1065–1078, 2004. `doi:10.1021/ci049958u`. 90

[FKP$^+$14] Fabrizio Frati, Michael Kaufmann, János Pach, Csaba D. Tóth, and David R. Wood. On the upward planarity of mixed plane graphs. *Journal of Graph Algorithms and Applications*, 18(2):253–279, 2014. `doi:10.7155/jgaa.00322`. 126

[FKŻ08a] Hanna Furmańczyk, Adrian Kosowski, and Paweł Żyliński. A note on mixed tree coloring. *Information Processing Letters*, 106(4):133–135, 2008. `doi:10.1016/j.ipl.2007.11.003`. 126

[FKŻ08b] Hanna Furmańczyk, Adrian Kosowski, and Paweł Żyliński. Scheduling with precedence constraints: Mixed graph coloring in series-parallel graphs. In *Proc. 7th International Conference on Parallel Processing and Applied Mathematics (PPAM'07)*, pages 1001–1008, 2008. `doi:10.1007/978-3-540-68111-3_106`. 125, 126

[FMM13] Stefan Felsner, George B. Mertzios, and Irina Mustata. On the recognition of four-directional orthogonal ray graphs. In *Proc. 38th International Symposium on Mathematical Foundations of Computer Science (MFCS'13)*, pages 373–384. Springer, 2013. (Some results herein are incomplete, see the warning in the full version: `http://page.math.tu-berlin.de/~felsner/Paper/dorgs.pdf`). `doi:10.1007/978-3-642-40313-2\_34`. 33

[FMM$^+$17] Stefan Funke, Thomas Mendel, Alexander Miller, Sabine Storandt, and Maria Wiebe. Map simplification with topology constraints: Exactly and in practice. In *Proc. 19th Workshop on Algorithm Engineering and Experiments (ALENEX'17)*, pages 185–196. SIAM, 2017. `doi:10.1137/1.9781611974768.15`. 213

[For87] Steven Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987. `doi:10.1007/BF01840357`. 188

[FR91] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software—Practice and Experience*, 21(11):1129–1164, 1991. `doi:10.1002/spe.4380211102`. 147

[Fra08] Fabrizio Frati. On minimum area planar upward drawings of directed trees and other families of directed acyclic graphs. *International Journal* *of Computational Geometry and Applications*, 18(03):251–271, 2008. doi:10.1142/S021819590800260X. 97, 99

[Gal68] Tibor Gallai. On directed graphs and circuits. In *Theory of Graphs (Proceedings of the Colloquium held at Tihany 1966)*, pages 115–118, 1968. 126

[Gei22] Joshua Geis. Aufwärtsplanare Zeichnungen mit drei Steigungen von Außenpfaden. Bachelor's thesis, Institut für Informatik, Universität Würzburg, 2022. (in German). URL: https://www1.pub.informatik.uni-wuerzburg.de/pub/theses/2022-geis-bachelor.pdf. 8, 116

[GGKS95] Paul W. Goldberg, Martin Charles Golumbic, Haim Kaplan, and Ron Shamir. Four strikes against physical mapping of DNA. *Journal of Computational Biology*, 2(1):139–152, 1995. doi:10.1089/cmb.1995.2.139. 136

[GHMS93] Leonidas J. Guibas, John Hershberger, Joseph S. B. Mitchell, and Jack Snoeyink. Approximating polygons and subdivisions with minimum link paths. *International Journal of Computational Geometry and Applications*, 3(4):383–415, 1993. doi:10.1142/S0218195993000257. 11, 188, 189, 190, 191, 194

[GHS07] Bernard Gendron, Alain Hertz, and Patrick St-Louis. On edge orienting methods for graph coloring. *Journal of Combinatorial Optimization*, 13:163–178, 2007. doi:10.1007/s10878-006-9019-3. 126

[GIP18] Daniel Gonçalves, Lucas Isenmann, and Claire Pennarun. Planar graphs as L-intersection or L-contact graphs. In *Proc. 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'18)*, pages 172–184. SIAM, 2018. URL: http://dl.acm.org/citation.cfm?id=3174304.3175278. 32

[GJ79] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979. 44

[GKK$^+$22] Ina Goeßmann, Jonathan Klawitter, Boris Klemz, Felix Klesen, Stephen G. Kobourov, Myroslav Kryven, Alexander Wolff, and Johannes Zink. The segment number: Algorithms and universal lower bounds for some classes of planar graphs. In *Proc. 48th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'22)*, pages 271–286. Springer, 2022. doi:10.1007/978-3-031-15914-5_20. 7, 58, 59

[GKNV93] Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Kiem-Phong Vo. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering*, 19(3):214–230, 1993. doi:10.1109/32.221135. 147

[GMA$^+$15] Mauricio G. Gruppi, Salles V. G. Magalhães, Marcus Vinícius Alvim Andrade, W. Randolph Franklin, and Wenli Li. An efficient and topologically correct map generalization heuristic. In *Proc. 17th International Conference on Enterprise Information Systems (ICEIS'15)*, pages 516–525, 2015. doi:10.5220/0005398105160525. 185

[GMR$^+$22] Grzegorz Gutowski, Florian Mittelstädt, Ignaz Rutter, Joachim Spoerhase, Alexander Wolff, and Johannes Zink. Coloring mixed and directional interval graphs. In *Proc. 30th International Symposium on Graph Drawing & Network Visualization (GD'22)*. Springer, 2022. doi:10.1007/978-3-031-22203-0_30. 9, 127, 128, 138

[God91] Michael Godau. A natural metric for curves – computing the distance for polygonal chains and approximation algorithms. In *Proc. 8th Annual Symposium on Theoretical Aspects of Computer Science (STACS'91)*, pages 127–136, 1991. doi:10.1007/BFb0020793. 11, 186, 187

[Gol80] Martin C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, 1980. 9, 126, 132

[GOT18] Jacob E. Goodman, Joseph O'Rourke, and Csaba D. Tóth, editors. *Handbook of Discrete and Computational Geometry*. Chapman and Hall/CRC, 3rd edition, 2018. URL: http://www.csun.edu/~ctoth/Handbook/HDCG3.html. 3

[Gro16] Martin Gronemann. Bitonic *st*-orderings for upward planar graphs. In *Proc. 24th International Symposium on Graph Drawing & Network Visualization (GD'16)*, pages 222–235. Springer, 2016. doi:10.1007/978-3-319-50106-2_18. 91

[GT01] Ashim Garg and Roberto Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM Journal on Computing*, 31(2):601–625, 2001. doi:10.1137/S0097539794277123. 24, 90, 119

[Hal93] Magnús M. Halldórsson. Approximating the minimum maximal independence number. *Information Processing Letters*, 46(4):169–172, 1993. doi:10.1016/0020-0190(93)90022-2. 216

[Har89] Richard I. Hartley. Drawing polygons given angle sequences. *Information Processing Letters*, 31(1):31–33, 1989. doi:10.1016/0020-0190(89)90105-1. 104

[Has65] Maria Hasse. Zur algebraischen Begründung der Graphentheorie. I. *Mathematische Nachrichten*, 28(5–6):275–290, 1965. (in German). doi:10.1002/mana.19650280503. 126

[HATI11] Bjarni V. Halldórsson, Derek Aguiar, Ryan Tarpine, and Sorin Istrail. The Clark phaseable sample size problem: Long-range phasing and loss of heterozygosity in GWAS. *Journal of Computational Biology*, 18(3):323–333, 2011. doi:10.1089/cmb.2010.0288. 33

[HBA$^+$18] Songtao He, Favyen Bastani, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, and Sam Madden. RoadRunner: improving the precision of road network inference from GPS trajectories. In *Proc. 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 3–12, 2018. doi:10.1145/3274895.3274974. 212

[HdL20] Martin Held and Stefan de Lorenzo. An efficient, practical algorithm and implementation for computing multiplicatively weighted voronoi diagrams. In *Proc. 28th Annual European Symposium on Algorithms (ESA'20)*, pages 56:1–56:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.56. 188

[HHE08] Weidong Huang, Seok-Hee Hong, and Peter Eades. Effects of crossing angles. In *Proc. 1st IEEE Pacific Visualization Symposium (PacificVis'08)*, pages 41–46. IEEE Computer Society, 2008. doi:10.1109/PACIFICVIS.2008.4475457. 3

[HKdW97] Pierre Hansen, Julio Kuplinsky, and Dominique de Werra. Mixed graph colorings. *Mathematical Methods of Operations Research*, 45:145–160, 1997. doi:10.1007/BF01194253. 9, 125, 126, 132

[HKMS18] Gregor Hültenschmidt, Philipp Kindermann, Wouter Meulemans, and André Schulz. Drawing planar graphs with few geometric primitives. *Journal of Graph Algorithms and Applications*, 22(2):357–387, 2018. doi:10.7155/jgaa.00473. 59, 60

[HL80] Bert Hölldobler and Charles J. Lumsden. Territorial strategies in ants. *Science*, 210(4471):732–739, 1980. doi:10.1126/science.210.4471.732. 1

[HL06] Patrick Healy and Karol Lynch. Two fixed-parameter tractable algorithms for testing upward planarity. *International Journal of Foundations of Computer Science*, 17(05):1095–1114, 2006. doi:10.1142/S0129054106004285. 91

[HNZ91] Irith Ben-Arroyo Hartman, Ilan Newman, and Ran Ziv. On grid intersection graphs. *Discrete Mathematics*, 87(1):41–52, 1991. doi:10.1016/0012-365X(91)90069-E. 33

[Hof17] Udo Hoffmann. On the complexity of the planar slope number problem. *Journal of Graph Algorithms and Applications*, 21(2):183–193, 2017. doi:10.7155/jgaa.00411. 8, 91, 123

[HS92] John Hershberger and Jack Snoeyink. Speeding up the Douglas-Peucker line-simplification algorithm. In *Proc. 5th International Symposium on Spatial Data Handling (SDH'92)*, pages 134–143, 1992. 188

[HS99] John Hershberger and Subhash Suri. An optimal algorithm for euclidean shortest paths in the plane. *SIAM Journal on Computing*, 28(6):2215–2256, 1999. `doi:10.1137/s0097539795289604`. 188

[HT74] John E. Hopcroft and Robert Endre Tarjan. Efficient planarity testing. *Journal of the ACM*, 21(4):549–568, 1974. `doi:10.1145/321850.321852`. 24

[II88] Hiroshi Imai and Masao Iri. Polygonal approximations of a curve – formulations and algorithms. In Godfried T. Toussaint, editor, *Computational Morphology*, volume 6 of *Machine Intelligence and Pattern Recognition*, pages 71–86. North-Holland, 1988. `doi:10.1016/B978-0-444-70467-2.50011-4`. 11, 186, 187, 190

[IMS17] Alexander Igamberdiev, Wouter Meulemans, and André Schulz. Drawing planar cubic 3-connected graphs with few segments: Algorithms & experiments. *Journal of Graph Algorithms and Applications*, 21(4):561–588, 2017. `doi:10.7155/jgaa.00430`. 58, 59

[Ise13] Tobias Isenberg. Visual abstraction and stylisation of maps. *The Cartographic Journal*, 50(1):8–18, 2013. `doi:10.1179/1743277412Y.0000000007`. 185

[JDG$^+$22] Iain G. Johnston, Kamaludin Dingle, Sam F. Greenbury, Chico Q. Camargo, Jonathan P. K. Doye, Sebastian E. Ahnert, and Ard A. Louis. Symmetry and simplicity spontaneously emerge from the algorithmic nature of evolution. *Proceedings of the National Academy of Sciences*, 119(11):e2113883119, 2022. `doi:10.1073/pnas.2113883119`. 1

[JJK$^+$13] Vít Jelínek, Eva Jelínková, Jan Kratochvíl, Bernard Lidický, Marek Tesař, and Tomáš Vyskočil. The planar slope number of planar partial 3-trees of bounded degree. *Graphs and Combinatorics*, 29(4):981–1005, 2013. `doi:10.1007/s00373-012-1157-z`. 91

[JLM98] Michael Jünger, Sebastian Leipert, and Petra Mutzel. Level planarity testing in linear time. In *Proc. 6th International Symposium on Graph Drawing (GD'98)*, pages 224–237. Springer, 1998. `doi:10.1007/3-540-37623-2\_17`. 33

[Joh74] David S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9(3):256–278, 1974. `doi:10.1016/S0022-0000(74)80044-9`. 229

[JSKC11] Jayadeva, Sameena Shah, Ravi Kothari, and Suresh Chandra. Debugging ants: How ants find the shortest route. In *Proc. 8th International Conference on Information, Communications & Signal Processing (ICICS'11)*, pages 1–5. IEEE, 2011. `doi:10.1109/ICICS.2011.6174275`. 1

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In *Proc. Symposium on the Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972. `doi:10.1007/978-1-4684-2001-2_9`. 125

[KHOO17] Ahmed Kouider, Hacène Ait Haddadène, Samia Ourari, and Ammar Oulamara. Mixed graph colouring for unit-time scheduling. *International Journal of Production Research*, 55(6):1720–1729, 2017. `doi:10.1080/00207543.2016.1224950`. 125

[kie] Kiel integrated environment for layout eclipse rich client (KIELER). (accessed on September 21, 2022). URL: `https://www.rtsys.informatik.uni-kiel.de/en/archive/kieler`. 143

[KKW15] Johannes Köbler, Sebastian Kuhnert, and Osamu Watanabe. Interval graph representation with given interval and intersection lengths. *Journal of Discrete Algorithms*, 34:108–117, 2015. `doi:10.1016/j.jda.2015.05.011`. 35

[KM94] Jan Kratochvíl and Jiří Matoušek. Intersection graphs of segments. *Journal of Combinatorial Theory, Series B*, 62(2):289–315, 1994. `doi:10.1006/jctb.1994.1071`. 32

[KM22] Jonathan Klawitter and Tamara Mchedlidze. Upward planar drawings with two slopes. *Journal of Graph Algorithms and Applications*, 26(1):171–198, 2022. `doi:10.7155/jgaa.00587`. 8, 24, 90, 92, 93, 94, 119

[KMLM16] Oh-Hyun Kwon, Chris Muelder, Kyungwon Lee, and Kwan-Liu Ma. A study of layout, rendering, and interaction methods for immersive graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 22(7):1802–1815, 2016. `doi:10.1109/TVCG.2016.2520921`. 3

[KMMS06] Dieter Kratsch, Ross M. McConnell, Kurt Mehlhorn, and Jeremy P. Spinrad. Certifying algorithms for recognizing interval graphs and permutation graphs. *SIAM Journal on Computing*, 36(2):326–353, 2006. `doi:10.1137/S0097539703437855`. 33

[KMS18] Philipp Kindermann, Wouter Meulemans, and André Schulz. Experimental analysis of the accessibility of drawings with few segments. *Journal of Graph Algorithms and Applications*, 22(3):501–518, 2018. `doi:10.7155/jgaa.00474`. 57

[KMSS19] Philipp Kindermann, Tamara Mchedlidze, Thomas Schneck, and Antonios Symvonis. Drawing planar graphs with few segments on a polynomial grid. In *Proc. 27th International Symposium on Graph Drawing & Network Visualization (GD'19)*, pages 416–429. Springer, 2019. doi:10.1007/978-3-030-35802-0_32. 58, 59

[KMW14] Kolja Knauer, Piotr Micek, and Bartosz Walczak. Outerplanar graph drawings with few slopes. *Computational Geometry*, 47(5):614–624, 2014. doi:10.1016/j.comgeo.2014.01.003. 91

[KOS19] Pavel Klavík, Yota Otachi, and Jirí Sejnoha. On the classes of interval graphs of limited nesting and count of lengths. *Algorithmica*, 81(4):1490–1511, 2019. doi:10.1007/s00453-018-0481-y. 35

[KPP13] Balázs Keszegh, János Pach, and Dömötör Pálvölgyi. Drawing planar graphs of bounded degree with few slopes. *SIAM Journal on Discrete Mathematics*, 27(2):1171–1183, 2013. doi:10.1137/100815001. 91

[KPPT08] Balázs Keszegh, János Pach, Dömötör Pálvölgyi, and Géza Tóth. Drawing cubic graphs with at most five slopes. *Computational Geometry*, 40(2):138–147, 2008. doi:10.1016/j.comgeo.2007.05.003. 91

[Kra91] Jan Kratochvíl. String graphs. II. recognizing string graphs is NP-hard. *Journal of Combinatorial Theory, Series B*, 52(1):67–78, 1991. doi:10.1016/0095-8956(91)90091-W. 32

[Kra94] Jan Kratochvíl. A special planar satisfiability problem and a consequence of its NP-completeness. *Discrete Applied Mathematics*, 52(3):233–252, 1994. doi:10.1016/0166-218X(94)90143-0. 33, 35

[Kra20] Rebecca Kraus. Level-außenplanare Zeichnungen mit wenigen Steigungen. Bachelor's thesis, Institut für Informatik, Universität Würzburg, 2020. (in German). 109

[KRW19] Myroslav Kryven, Alexander Ravsky, and Alexander Wolff. Drawing graphs on few circles and few spheres. *Journal of Graph Algorithms and Applications*, 23(2):371–391, 2019. doi:10.7155/jgaa.00495. 60

[KZ23] Jonathan Klawitter and Johannes Zink. Upward planar drawings with three and more slopes. *Journal of Graph Algorithms and Applications*, 27(2):49–70, 2023. doi:10.7155/jgaa.00617. 8

[Li97] Wing Ning Li. Two-segmented channel routing is strong NP-complete. *Discrete Applied Mathematics*, 78(1–3):291–298, 1997. doi:10.1016/S0166-218X(97)00020-6. 48

[LLMN13] William Lenhart, Giuseppe Liotta, Debajyoti Mondal, and Rahnuma Islam Nishat. Planar and plane slope number of partial 2-trees. In *Proc. 21st International Symposium on Graph Drawing (GD'13)*, pages 412–423. Springer, 2013. doi:10.1007/978-3-319-03841-4_36. 91

[LWZ16] Fabian Lipp, Alexander Wolff, and Johannes Zink. Faster force-directed graph drawing with the well-separated pair decomposition. *Algorithms*, 9(3):53, 2016. doi:10.3390/a9030053. 147

[Mat14] Jiří Matoušek. Intersection graphs of segments and $\exists\mathbb{R}$. arXiv preprint, 2014. arXiv:1406.2636. 16, 32

[MCW$^+$19] Yang Min, Cailian Chen, Xiaoyu Wang, Jianping He, and Yang Zhang. SGM: Seed growing map-matching with trajectory fitting. In *Proc. 5th International Conference on Big Data Computing and Communications (BIGCOM'19)*, pages 204–212. IEEE, 2019. doi:10.1109/bigcom.2019.00036. 185

[MdB04] Nirvana Meratnia and Rolf A. de By. Spatiotemporal compression techniques for moving point objects. In *Proc. 9th International Conference on Extending Database Technology (EDBT'04)*, pages 765–782. Springer, 2004. doi:10.1007/978-3-540-24741-8_44. 185

[MHS$^+$22] Stuart McAlpine, John C. Helly, Matthieu Schaller, Till Sawala, Guilhem Lavaux, Jens Jasche, Carlos S. Frenk, Adrian Jenkins, John R. Lucey, and Peter H. Johansson. SIBELIUS-DARK: a galaxy catalogue of the local volume from a constrained realization simulation. *Monthly Notices of the Royal Astronomical Society*, 512(4):5823–5847, 2022. doi:10.1093/mnras/stac295. 1

[MM99] Terry A. McKee and Fred R. McMorris. *Topics in Intersection Graph Theory*. Discrete Mathematics and Applications. SIAM, 1999. doi:10.1137/1.9780898719802. 31

[MMP87] Joseph S. B. Mitchell, David M. Mount, and Christos H. Papadimitriou. The discrete geodesic problem. *SIAM Journal on Computing*, 16(4):647–668, 1987. doi:10.1137/0216045. 188

[MNBR13] Debajyoti Mondal, Rahnuma Islam Nishat, Sudip Biswas, and Md. Saidur Rahman. Minimum-segment convex drawings of 3-connected cubic plane graphs. *Journal of Combinatorial Optimization*, 25(3):460–480, 2013. doi:10.1007/s10878-011-9390-6. 58, 59

[MO88] Avraham Melkman and Joseph O'Rourke. On polygonal chain approximation. In Godfried T. Toussaint, editor, *Computational Morphology*, volume 6 of *Machine Intelligence and Pattern Recognition*, pages 87–95. North-Holland, 1988. doi:10.1016/B978-0-444-70467-2.50012-6. 11, 186, 189, 190, 191, 194, 205

[MP11] Padmini Mukkamala and Dömötör Pálvölgyi. Drawing cubic graphs with the four basic slopes. In *Proc. 19th International Symposium on Graph Drawing (GD'11)*, pages 254–265. Springer, 2011. doi:10.1007/978-3-642-25878-7_25. 91

[MS07] Matthias Müller-Hannemann and Anna Schulze. Hardness and approximation of octilinear steiner trees. *International Journal of Computational Geometry and Applications*, 17(3):231–260, 2007. `doi:10.1142/S021819590700232X`. 90

[MS09] Padmini Mukkamala and Mario Szegedy. Geometric representation of cubic graphs with four directions. *Computational Geometry*, 42(9):842–851, 2009. `doi:10.1016/j.comgeo.2009.01.005`. 91

[NGM$^+$07] Viet Nguyen, Stefan Gächter, Agostino Martinelli, Nicola Tomatis, and Roland Siegwart. A comparison of line extraction algorithms using 2D range data for indoor mobile robotics. *Autonomous Robots*, 23(2):97–111, 2007. `doi:10.1007/s10514-007-9034-y`. 185

[NW11] Martin Nöllenburg and Alexander Wolff. Drawing and labeling high-quality metro maps by mixed-integer programming. *IEEE Transactions on Visualization and Computer Graphics*, 17(5):626–641, 2011. `doi:10.1109/TVCG.2010.81`. 90

[Nö05] Martin Nöllenburg. Automated drawing of metro maps. Diploma thesis, University of Karlsruhe (TH), 2005. URL: `https://i11www.iti.kit.edu/extra/publications/n-admm-05da.pdf`. 109

[ODB21] Alihan Okka, Ugur Dogrusoz, and Hasan Balci. CoSEP: A compound spring embedder layout algorithm with support for ports. *Information Visualization*, 20(2–3):151—169, 2021. `doi:10.1177/14738716211028136`. 143

[ORW19] Yoshio Okamoto, Alexander Ravsky, and Alexander Wolff. Variants of the segment number of a graph. In *Proc. 27th International Symposium on Graph Drawing & Network Visualization (GD'19)*, pages 430–443. Springer, 2019. `doi:10.1007/978-3-030-35802-0_33`. 59

[Pap94] Achilleas Papakostas. Upward planarity testing of outerplanar DAGs. In *Proc. DIMACS International Workshop on Graph Drawing (GD'94)*, pages 298–306, 1994. `doi:10.1007/3-540-58950-3_385`. 91, 108

[PCA02] Helen C. Purchase, David A. Carrington, and Jo-Anne Allder. Empirical evaluation of aesthetics-based graph layout. *Empirical Software Engineering*, 7(3):233–255, 2002. `doi:10.1023/A:1016344215610`. 3, 168

[PHNK12] Helen C. Purchase, John Hamer, Martin Nöllenburg, and Stephen G. Kobourov. On the usability of Lombardi graph drawings. In *Proc. 20th International Symposium on Graph Drawing (GD'12)*, pages 451–462. Springer, 2012. `doi:10.1007/978-3-642-36763-2_40`. 3, 60

[PMCC01] Helen C. Purchase, Matthew McGill, Linda Colpoys, and David A. Carrington. Graph drawing aesthetics and the comprehension of UML class diagrams: An empirical study. In *Proc. Australian Symposium on Information Visualisation (invis.au'01)*, pages 129–137. Australian Computer Society, 2001. URL: `http://crpit.scem.westernsydney.edu.au/abstracts/CRPITV9Purchase2.html`. 3

[PP06] János Pach and Dömötör Pálvölgyi. Bounded-degree graphs can have arbitrarily large slope numbers. *Electronic Journal of Combinatorics*, 13(1):N1, 2006. `doi:10.37236/1139`. 91

[Pra77] Vaughan R. Pratt. Two easy theories whose combination is hard. Technical report, Massachusetts Institute of Technology, 1977. URL: `http://boole.stanford.edu/pub/sefnp.pdf`. 54

[pra20a] PRALINE data structure and layouting algorithm, 2020. (accessed on September 21, 2022). URL: `https://github.com/j-zink-wuerzburg/praline`. 156

[pra20b] PRALINE pseudo plans – algorithm and data sets, 2020. (accessed on September 21, 2022). URL: `https://github.com/j-zink-wuerzburg/pseudo-praline-plan-generation`. 156, 158

[PS97] Itsik Pe'er and Ron Shamir. Realizing interval graphs with size and distance constraints. *SIAM Journal on Discrete Mathematics*, 10(4):662–687, 1997. `doi:10.1137/S0895480196306373`. 35

[PSD09] Mathias Pohl, Markus Schmitt, and Stephan Diehl. Comparing the readability of graph layouts using eyetracking and task-oriented analysis. In *Proc. International Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging (CAe'09)*, pages 49–56. Eurographics Association, 2009. `doi:10.2312/COMPAESTH/COMPAESTH09/049-056`. 3

[Qua21] Valentin Quapil. Upward and upward-planar drawings with limited slopes. Bachelor's Thesis, Karlsruhe Institute of Technology, 2021. URL: `https://i11www.iti.kit.edu/_media/teaching/theses/ba-quapil-21.pdf`. 8, 92, 99, 100, 123

[Ram72] Urs Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1(3):244–256, 1972. `doi:10.1016/S0146-664X(72)80017-0`. 185, 188

[RdW08] Bernard Ries and Dominique de Werra. On two coloring problems in mixed graphs. *European Journal of Combinatorics*, 29(3):712–725, 2008. `doi:10.1016/j.ejc.2007.03.006`. 126

[RFO$^+$24] Yan Alves Radtke, Stefan Felsner, Johannes Obenaus, Sandro Roch, Manfred Scheucher, and Birgit Vogtenhuber. Flip graph connectivity for arrangements of pseudolines and pseudocircles. In *Proc. 35th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'24)*, pages 4849–4871. SIAM, 2024. `doi:10.1137/1.9781611977912.172`. 61

[RLP$^+$15] Chris R. Reid, Matthew J. Lutz, Scott Powell, Albert B. Kao, Iain D. Couzin, and Simon Garnier. Army ants dynamically adjust living bridges in response to a cost–benefit trade-off. *Proceedings of the National Academy of Sciences*, 112(49):15113–15118, 2015. `doi:10.1073/pnas.1512241112`. 1

[Rob70] Fred S. Roberts. On nontransitive indifference. *Journal of Mathematical Psychology*, 7(2):243–258, 1970. `doi:10.1016/0022-2496(70)90047-7`. 136

[Roy67] B. Roy. Nombre chromatique et plus longs chemins d'un graphe. *Revue Française d'Informatique Recherche Opérationnelle*, 1(5):129–132, 1967. (in French). `doi:10.1051/m2an/1967010501291`. 126

[Rus20] Irena Rusu. Stick graphs: examples and counter-examples. arXiv preprint, 2020. `arXiv:2007.10773`. 35

[Rus22] Irena Rusu. Forced pairs in A-stick graphs. *Discrete Mathematics*, 345(9), 2022. `doi:10.1016/j.disc.2022.112962`. 6, 35

[Rus23] Irena Rusu. On the complexity of recognizing stick, BipHook and max point-tolerance graphs. *Theoretical Computer Science*, 952:113773, 2023. `doi:10.1016/J.TCS.2023.113773`. 6, 35

[SAAR08] Md. Abul Hassan Samee, Md. Jawaherul Alam, Muhammad Abdullah Adnan, and Md. Saidur Rahman. Minimum segment drawings of series-parallel graphs with the maximum degree three. In *Proc. 16th International Symposium on Graph Drawing (GD'08)*, pages 408–419. Springer, 2008. `doi:10.1007/978-3-642-00219-9_40`. 59

[San94] Georg Sander. Graph layout through the VCG tool. In *Proc. 2nd DIMACS International Workshop on Graph Drawing (GD'94)*, pages 194–205. Springer, 1994. `doi:10.1007/3-540-58950-3_371`. 151

[SBS87] Jeremy Spinrad, Andreas Brandstädt, and Lorna Stewart. Bipartite permutation graphs. *Discrete Applied Mathematics*, 18(3):279–292, 1987. `doi:10.1016/S0166-218X(87)80003-3`. 33, 36

[Sch90] Walter Schnyder. Embedding planar graphs on the grid. In *Proc. 1st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'90)*, pages 138–148. SIAM, 1990. URL: `https://dl.acm.org/doi/10.5555/320176.320191`. 24

[Sch09] Marcus Schaefer. Complexity of some geometric and topological problems. In *Proc. 17th International Symposium on Graph Drawing (GD'09)*, pages 334–344. Springer, 2009. doi:10.1007/978-3-642-11805-0_32. 16

[Sch15] André Schulz. Drawing graphs with few arcs. *Journal of Graph Algorithms and Applications*, 19(1):393–412, 2015. doi:10.7155/jgaa.00366. 7, 57, 60, 89

[SN22] Jonathan Y. Suen and Saket Navlakha. A feedback control principle common to several biological and engineered systems. *Journal of The Royal Society Interface*, 19(188), 2022. doi:10.1098/rsif.2021.0711. 1

[Sot20] Yuri N. Sotskov. Mixed graph colorings: A historical review. *Mathematics*, 8(3):385:1–24, 2020. doi:10.3390/math8030385. 125

[SS94] Malay K. Sen and Barun K. Sanyal. Indifference digraphs: A generalization of indifference graphs and semiorders. *SIAM Journal on Discrete Mathematics*, 7(2):157–165, 1994. doi:10.1137/S0895480190177145. 33

[SSS03] Marcus Schaefer, Eric Sedgwick, and Daniel Stefankovic. Recognizing string graphs in NP. *J. Comput. Syst. Sci.*, 67(2):365–380, 2003. doi:10.1016/S0022-0000(03)00045-X. 32

[SSvH14] Christoph Daniel Schulze, Miro Spönemann, and Reinhard von Hanxleden. Drawing layered graphs with port constraints. *Journal of Visual Languages and Computing*, 25(2):89–106, 2014. doi:10.1016/j.jvlc.2013.11.005. 10, 143, 148, 150

[SSZ20] Joachim Spoerhase, Sabine Storandt, and Johannes Zink. Simplification of polyline bundles. In *Proc. 17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT'20)*, pages 35:1–35:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.SWAT.2020.35. 13

[SSZ23] Peter Schäfer, Sabine Storandt, and Johannes Zink. Optimal polyline simplification under the local Fréchet distance in (near-)quadratic time. In *Proc. 35th Canadian Conference on Computational Geometry (CCCG'23)*, pages 225–238, 2023. URL: https://wadscccg2023.encs.concordia.ca/assets/pdf/CCCG_2023_proc.pdf#section.0.26, arXiv:2201.01344. 12

[ST76] Yuri N. Sotskov and Vyacheslav S. Tanaev. Chromatic polynomial of a mixed graph. *Vestsi Akademii Navuk BSSR. Seryya Fizika-Matematychnykh Navuk*, 6:20–23, 1976. (In Russian). 125

[Ste51] S. K. Stein. Convex maps. *Proceedings of the American Mathematical Society*, 2(3):464–466, 1951. 24

[STT81] Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981. `doi:10.1109/TSMC.1981.4308636`. 10, 89, 127, 142, 145, 149

[STTU11] Anish Man Singh Shrestha, Asahi Takaoka, Satoshi Tayu, and Shuichi Ueno. On two problems of nano-PLA design. *IEICE Transactions*, 94-D(1):35–41, 2011. `doi:10.1587/transinf.E94.D.35`. 33

[Szp45] Edward Szpilrajn-Marczewski. Sur deux propriétés des classes d'ensembles. *Fundamenta Mathematicae*, 33(1):303–307, 1945. (in French). URL: `http://eudml.org/doc/213098`. 31

[Tam13] Roberto Tamassia, editor. *Handbook of Graph Drawing and Visualization.* Chapman and Hall/CRC, 2013. URL: `https://cs.brown.edu/people/rtamassi/gdhandbook/`. 15

[Tar72] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. `doi:10.1137/0201010`. 19, 85

[vdKKL$^+$19] Mees van de Kerkhof, Irina Kostitsyna, Maarten Löffler, Majid Mirzanezhad, and Carola Wenk. Global curve simplification. In *Proc. 27th Annual European Symposium on Algorithms (ESA'19)*, pages 67:1–67:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.ESA.2019.67`. 188, 213

[Vit62] L. M. Vitaver. Determination of minimal coloring of vertices of a graph by means of boolean powers of the incidence matrix. *Doklady Akademii Nauk SSSR*, 147:758–759, 1962. (in Russian). URL: `https://www.ams.org/mathscinet-getitem?mr=0145509`. 126

[vKLW20] Marc J. van Kreveld, Maarten Löffler, and Lionov Wiratma. On optimal polyline simplification using the Hausdorff and Fréchet distance. *Journal of Computational Geometry*, 11(1):1–25, 2020. `doi:10.20382/jocg.v11i1a1`. 185, 186, 188, 209

[VW90] Mahes Visvalingam and J. Duncan Whyatt. The Douglas-Peucker algorithm for line simplification: Re-evaluation through visualization. *Computer Graphics Forum*, 9(3):213–228, 1990. `doi:10.1111/j.1467-8659.1990.tb00398.x`. 185

[VW93] Mahes Visvalingam and J. Duncan Whyatt. Line generalisation by repeated elimination of points. *The Cartographic Journal*, 30(1):46–51, 1993. `doi:10.1179/000870493786962263`. 188

[Wag36] Klaus Wagner. Bemerkungen zum Vierfarbenproblem. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 46:26–32, 1936. (in German). 24

[WC94] Greg A. Wade and Jiang-Hsing Chu. Drawability of complete graphs using a minimal slope set. *The Computer Journal*, 37(2):139–142, 1994. `doi:10.1093/comjnl/37.2.139`. 91

[wik17] Wikimedia Commons contributors. Resistance thermometer, 2017. (accessed on 2022/10/13). URL: `https://upload.wikimedia.org/wikipedia/commons/b/b0/Hexanitroazobenzene_structure.svg`. 90

[wik21] Wikimedia Commons contributors. Structure of hexaazobenzene, 2021. (accessed on 2022/10/13). URL: `https://upload.wikimedia.org/wikipedia/commons/b/b0/Hexanitroazobenzene_structure.svg`. 90

[wik22a] Wikipedia contributors. Ant colony optimization algorithms – Wikipedia, The Free Encyclopedia, 2022. (accessed on 2022/12/18). URL: `https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms`. 1

[wik22b] Wikipedia contributors. Combinatorial explosion – Wikipedia, The Free Encyclopedia, 2022. (accessed on 2022/12/19). URL: `https://en.wikipedia.org/wiki/Combinatorial_explosion`. 4

[wik22c] Wikipedia contributors. Library sort – Wikipedia, The Free Encyclopedia, 2022. (accessed on 2022/12/18). URL: `https://en.wikipedia.org/wiki/Library_sort`. 1

[wik22d] Wikipedia contributors. List of NP-complete problems – Wikipedia, The Free Encyclopedia, 2022. (accessed on 2022/12/18). URL: `https://en.wikipedia.org/wiki/List_of_NP-complete_problems`. 2

[WM03] Shin-Ting Wu and Mercedes Rocío Gonzales Márquez. A non-self-intersection Douglas-Peucker algorithm. In *Proc. 16th Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI'03)*, pages 60–66. IEEE Computer Society, 2003. `doi:10.1109/sibgra.2003.1240992`. 185

[WPCM02] Colin Ware, Helen C. Purchase, Linda Colpoys, and Matthew McGill. Cognitive measurements of graph aesthetics. *Information Visualization*, 1(2):103–110, 2002. `doi:10.1057/palgrave.ivs.9500013`. 3, 168

[WS75] William A. Wagenaar and Sabato D. Sagaria. Misperception of exponential growth. *Perception & Psychophysics*, 18:416–422, 1975. `doi:10.3758/BF03204114`. 4

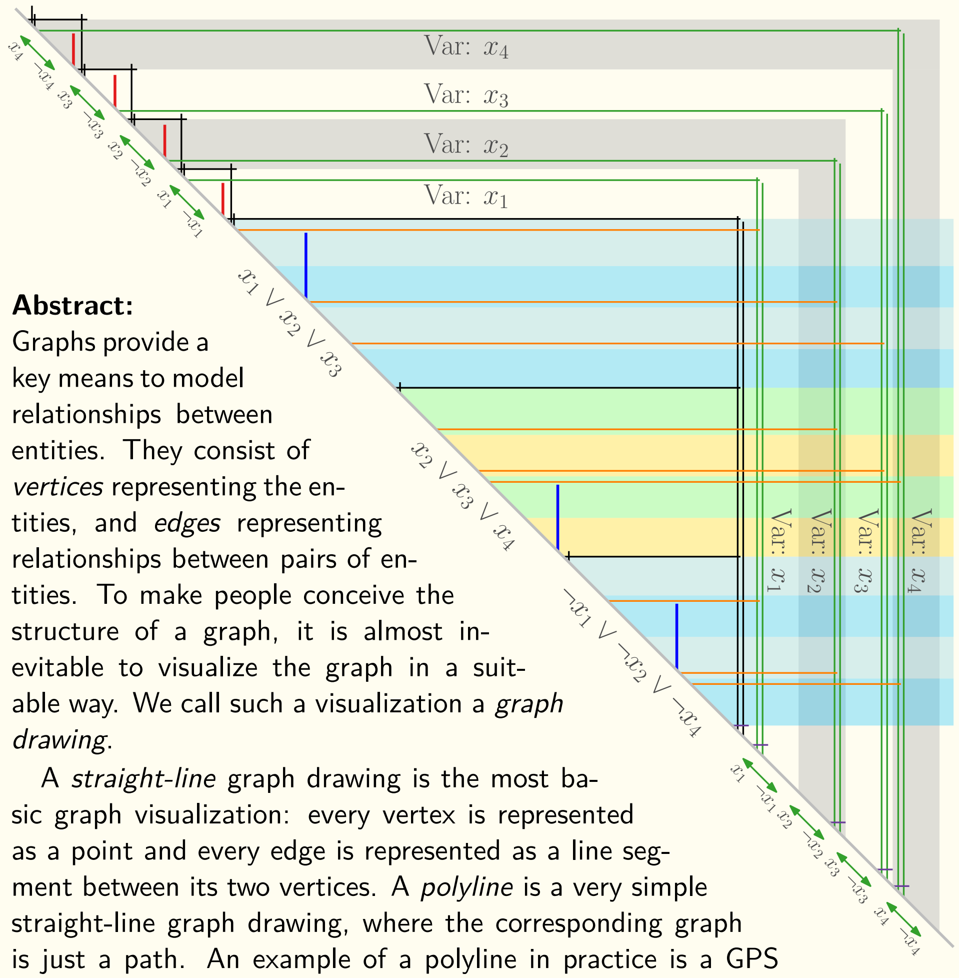
[wvv22] Würzburger Versorgungs und Verkehrs-GmbH (WVV). Liniennetz Straßenbahn Würzburg, 2022. (accessed on 2022/10/13). URL: `https://www.wvv.de/media-wvv/mobilitaet/dokumente/liniennetz/liniennetz-strassenbahn-wuerzburg.pdf`. 90

[WZ13] Tim Wylie and Binhai Zhu. Protein chain pair simplification under the discrete Fréchet distance. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 10(6):1372–1383, 2013. `doi:10.1109/tcbb.2013.17`. 212

[WZBW20] Julian Walter, Johannes Zink, Joachim Baumeister, and Alexander Wolff. Layered drawing of undirected graphs with generalized port constraints. In *Proc. 28th International Symposium on Graph Drawing & Network Visualization (GD'20)*, pages 220–234. Springer, 2020. `doi:10.1007/978-3-030-68766-3_18`. 167

[XRPH12] Kai Xu, Chris Rooney, Peter J. Passmore, and Dong-Han Ham. A user study on curved edges in graph visualisation. In *Proc. 7th International Conference on the Theory and Application of Diagrams (DIAGRAMS'10)*, pages 306–308. Springer, 2012. `doi:10.1007/978-3-642-31223-6_34`. 60

[ZWBW22] Johannes Zink, Julian Walter, Joachim Baumeister, and Alexander Wolff. Layered drawing of undirected graphs with generalized port constraints. *Computational Geometry*, 105–106:101886, 2022. `doi:10.1016/j.comgeo.2022.101886`. 10



Var: x_4
Var: x_3
Var: x_2
Var: x_1
x_4 $\neg x_4$ x_3 $\neg x_3$ x_2 $\neg x_2$ x_1 $\neg x_1$
$x_1 \vee x_2 \vee x_3$
$x_2 \vee x_3 \vee x_4$
$\neg x_1 \vee \neg x_2 \vee \neg x_4$
Var: x_1
Var: x_2
Var: x_3
Var: x_4
x_1 $\neg x_1$ x_2 $\neg x_2$ x_3 $\neg x_3$ x_4 $\neg x_4$

Abstract:
Graphs provide a key means to model relationships between entities. They consist of *vertices* representing the entities, and *edges* representing relationships between pairs of entities. To make people conceive the structure of a graph, it is almost inevitable to visualize the graph in a suitable way. We call such a visualization a *graph drawing*.

A *straight-line* graph drawing is the most basic graph visualization: every vertex is represented as a point and every edge is represented as a line segment between its two vertices. A *polyline* is a very simple straight-line graph drawing, where the corresponding graph is just a path. An example of a polyline in practice is a GPS trajectory. The underlying road network, in turn, can be modeled as a graph.

Although line segments possess a very simple structure, many interesting theoretical problems arise when using them in the context of graph and polyline drawings. We investigate some of these problems in this book. In particular, we study algorithms for recognizing certain graphs representable with line segments, for generating straight-line graph drawings, and for abstracting polylines. Besides theoretical results, we consider applications for such algorithms like automatic layouting of cable plans.

The illustrations on the cover pages show two constructions of NP-hardness proofs given in this book. The one on the back page is based on *stick graphs*, that is, graphs that can be realized as vertical and horizontal line segments drawn onto a line of slope -1. The one on the front page sketches a so-called *outerplanar* graph drawing. The line segment there shall represent many tiny squares. Each such square consists of five edges that are drawn with line segments of only three distinct slopes.