Some Theoretical Perspectives on Recent Challenges in Graph Drawing

Dissertation
der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Axel Kuckuk
aus Balingen

Tübingen
2023

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	14.06.2024
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter/-in:	Prof. Dr. Michael Kaufmann
2. Berichterstatter/-in:	Prof. Dr. Klaus-Jörn Lange

Abstract

Graph Drawing provides essential tools to visualize complex and extensive systems. In doing so it allows humans to see, learn and understand what would otherwise remain unseen, unlearned and incomprehensible. As new areas of application arise regularly, ranging from phylogenetic trees, displaying the evolutionary history of all life on earth in biology, to flowcharts, visually representing processes of virtually everything in schools, universities and business offices all over the world, it is unsurprising that new challenges emerge constantly. In this thesis we will investigate three recent challenges emanating from practical applications, bringing them into the theoretical context and provide applicable solutions.

The first problem concerns the representation of bipartite graphs whose vertex set consists of two disjoint parts. Commonly bipartite graphs are displayed using 2-layer drawings, where each part is drawn on a distinct horizontal line. However, unless the graph is relatively small, the representation does not fit on a single computer screen. While this is inevitable, we would like to make sure one vertex with all its neighbors still fits on a single screen. Typically the two parts of the vertex set play considerably different roles in the data set, which leads to different variants of the problem. This introduces the so-called Window Width problem, where vertices are rearranged such that the smallest screen possible suffices. We prove that one variant of the problem is polynomial time solvable, while another one is NP-hard. Additionally, we provide an approximation algorithm for the second variant and study relaxed problems. Further, some experiments are performed, evaluating the practical impact of the implemented algorithms.

The second problem considers the display of multiple hierarchical structures at once. Particularly, we investigate into the simultaneous embedding of multiple trees, where the visualization requires a specific order of the leaf vertices. Such a requirement is useful for many application, as it allows to show the data points sorted, for instance alphabetically, to enable efficient lookup. Specifically, k rooted, layered trees are provided whose leaves are all placed on the same layer, in a required total order. The problem consists of finding orders for the vertices of all other layers, such that the total number of crossings is minimal. While it is known that this problem is NP-hard for arbitrarily many trees even on only two layers, we provide an FPL-algorithm in k if the number of layers is two, and an XP-algorithm in k if the number of layers is three. For arbitrarily many layers we show that the problem is polynomial time solvable when restricting the number of trees to two.

The third problem considers dynamic visualizations. While static visualizations are still relevant for technical and analogue applications, the general public increasingly uses interactive visualizations. Thus, it is necessary to make known drawing methods applicable for dynamic tasks. Specifically we consider orthoradial metro maps. These representations of public transportation networks show the lines of transportation along concentric circles and on lines emanating from the center point. We extend the drawing model to be suitable for dynamic applications, proposing drawing strategies which are applicable for continuously updating and geospatially augmented data. Further we introduce a hybrid visualization model combining locally precise and globally schematic representations. Finally, we briefly discuss five other recent challenges for which we provide well-readable visualizations, before concluding the thesis with some interesting open research problems.

Zusammenfassung

Graphenzeichnen stellt essenzielle Methoden zur Verfügung, um komplexe und umfangreiche Systeme zu visualisieren. Dies ermöglicht dem Menschen, Dinge zu sehen, zu lernen und zu verstehen, die ansonsten ungesehen, ungelernt und unverstanden blieben. Da zunehmend neue Anwendungsbereiche entstehen, von phylogenetischen Bäumen, die in der Biologie die Evolutionsgeschichte allen Lebens darstellen, bis hin zu Flussdiagrammen, die in Schulen, Universitäten und in Unternehmen weltweit Abläufe aller Art visualisieren, ist es nicht überraschend, dass sich uns laufend neue Herausforderungen stellen. In dieser Arbeit werden wir drei dieser aktuellen Herausforderungen aus der praktischen Anwendung untersuchen, sie in einen theoretischen Kontext stellen und anwendbare Lösungen bieten.

Das erste Problem behandelt die Darstellung von zweistufigen Graphen, deren Knotenmenge aus zwei disjunkten Teilen besteht. Üblicherweise werden zweistufige Graphen mithilfe von 2-Layer-Zeichnungen dargestellt, bei denen jeder der zwei Teile auf einer eigenen horizontalen Linie gezeichnet wird. Abgesehen von ausgesprochen kleinen Graphen passt die Darstellung in aller Regel nicht auf einen einzigen Computerbildschirm. Dies ist zwar unvermeidlich, allerdings können wir versuchen sicherzustellen, dass jeder Knoten mit all seinen Nachbarknoten vollständig auf einen einzigen Bildschirm passt. Diese Aufgabenstellung bezeichnen wir als das Window Width Problem, von dem wir mehrere Varianten untersuchen werden. Dabei sollen die Knoten so angeordnet werden, dass die Bedingung auch für einen kleinstmöglichen Bildschirm erfüllt ist. Wir beweisen, dass eine Variante des Problems in Polynomialzeit lösbar ist, während eine andere Variante NP-schwer ist. Außerdem wird ein Approximationsalgorithmus für die zweite Variante entwickelt und relaxierte Probleme untersucht. Darüber hinaus werden einige Experimente durchgeführt, um den praktischen Wirkungsgrad der implementierten Algorithmen zu bewerten.

Das zweite Problem widmet sich der gleichzeitigen Darstellung mehrerer hierarchischer Strukturen. Insbesondere untersuchen wir die gleichzeitige Einbettung von mehreren Bäumen, bei denen die Visualisierung eine bestimmte Reihenfolge der Blattpunkte erfordert. Eine solche Bedingung ist für viele Anwendungen nützlich, da sie es ermöglicht, die Datenpunkte sortiert darzustellen, zum Beispiel alphabetisch, um ein effizientes Auffinden zu bewirken. Konkret werden k verwurzelte, gestufte Bäume vorgegeben, deren Blätter sich alle auf derselben Ebene befinden, und zwar in einer geforderten Reihenfolge. Das Problem besteht darin, Ordnungen für die Knoten aller anderen Stufen zu finden, sodass die Gesamtzahl der Kreuzungen minimal ist. Es ist zwar bekannt, dass dieses Problem für beliebig viele Bäume auch auf nur zwei Stufen NP-schwer ist, aber wir liefern einen

FPL-Algorithmus in k, für den Fall mit zwei Stufen und einen XP-Algorithmus in k für den Fall mit drei Stufen. Für beliebig viele Schichten zeigen wir, dass das Problem in Polynomialzeit lösbar ist, wenn man die Anzahl der Bäume auf zwei beschränkt.
Das dritte Problem behandelt dynamische Visualisierungen. Während statische Darstellungen für technische und analoge Anwendungen noch immer relevant sind, nutzt die breite Öffentlichkeit zunehmend interaktive Visualisierungen. Daher ist es notwendig, bekannte Zeichenmethoden für dynamische Aufgaben nutzbar zu machen. Konkret betrachten wir ortho-radiale Metrokarten. Diese Abbildungen öffentlicher Verkehrsnetze zeigen die Verkehrslinien entlang konzentrischer Kreise und auf Linien, die vom Mittelpunkt ausgehen. Wir passen das Zeichnungsmodell so an, dass es sich für dynamische Anwendungen eignet, und schlagen konkrete Zeichenstrategien vor, die für kontinuierlich aktualisierende und für georäumliche Netzwerke nutzbar sind. Darüber hinaus stellen wir ein hybrides Visualisierungsmodell vor, das lokal exakte und global schematische Darstellungen kombiniert.
Zuletzt widmen wir uns knapp fünf weiteren aktuellen Herausforderungen, für welche wir gut lesbare Visualisierungen präsentieren, bevor wir die Arbeit mit einigen interessanten offenen Forschungsproblemen abschließen.

Acknowledgement

First of all, I would like to thank Michael Kaufmann, who opened up the most exciting aspect of Computer Science to me and gave me the opportunity to write this thesis as part of his working group.

I am also indebted to my other co-authors, Maximilian Pfister, Michalis Bekos, Henry Förster, Stephen Kobourov, Miroslav Kryven, Julia Katheder, Lena Schlipf and Johannes Zink. Michalis and Henry in particular have helped me a lot, showing me the ropes of doing scientific work.

I would also like to thank my current and past colleagues Patrizio Angelini, Michael Bekos, Henry Förster, Renate Hallmayer, Maxi Pfister, Lena Schlipf and Thomas Schneck. Perhaps even more than the research field, you were the reason why I stayed in science for so long.

I would especially like to emphasize the invaluable support of Lena Schlipf, Henry Förster and Maxi Pfister in the form of deeply necessary and amazingly fast test reading.

Of course, I would also like to thank my parents for their constant support, even though I have so far been unable to convince them that Graph Drawing has any real-world effects.

Finally, and most of all, my deepest thanks go to Sarah, without whom I would never have made it to this point (or any other point, to be frank).

Contents

Chapter 1

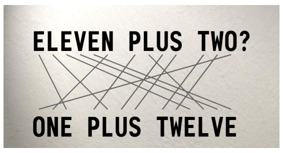
Introduction

Graph Drawing is a research area that lies at the intersection of Computer Science and Mathematics. It combines methods from information visualization and geometric graph theory to provide (usually two-dimensional) geometric representations of graphs. As humans rely heavily on visual perception, with the visual cortex being the largest system in the human brain [131], a demand for network visualization arises from many applications. Some examples among countless others are: Flowcharts [64] (which receive increasing popularity since 1921), Sociograms [123] (representing social networks), State diagrams [18] (visualizing finite state machines), Data-flow diagrams [138] (showing software-system processes) and visual analytics [130] (enabling human-information discourse by interactive visual interfaces).
As Graph Drawing methods provide the elementary tools for network visualization, a particular focus is laid on the interworking of implementable algorithms and theoretical advances.

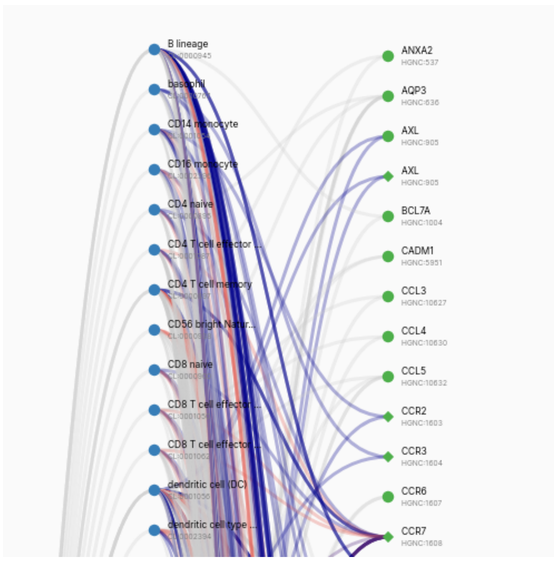
For the same graph there exist widely different visual representations, however, the properties of a specific visualization, characterized by the assignment of vertices to coordinates and the representation of edges as curves connecting their respective vertices, influence the readability [128], usability [106], aesthetics [107] but also realizability [118] and fabrication cost [40, 79].

While fundamental drawing styles and properties such as area [42, 55, 82], bends [36, 120] and crossings [67, 81, 92] have been extensively studied for many graph classes, with the area of application being ever-evolving, adapting to the technological advance and including continuously more fields, the demand for theoretical advances follows suit.

Thus, in this work we will highlight exactly the point where demands by practical applications and theoretical solutions meet and share some theoretical perspectives on recent challenges. The thesis consists of three parts, each representative of a major branch of Graph Drawing. For each part a theoretical problem will arise through challenges encountered in applications of network visualization and several new results will be derived to provide a solution to the problem. To this end, we introduce the settings encountered in this thesis in order of appearance.



(a) A 2-layer drawing representing an anagram, generated by dict.cc/shuffle [73].



(b) Picture detail of a state-of-the-art visualization tool for anatomical structures, taken from [34]. Blood cell types are shown on the left and biomarkers on the right. This yields a 2-layer drawing, however rotated by 90°. Note that only a small section of the graph fits on the screen.

Figure 1.1

2-Layer. A well studied class of graphs are the *hierarchical* graphs, where there exists an affiliation relationship between objects of the data. For instance, hierarchical graphs appear in Genealogy, where any gene has one unique predecessor [2], in Biology, where organs consist of specific cells, which in turn can contain markers [20], or in source code, where any code block can be part of a 'parent' code block [137].

A common way to represent this hierarchical information is to separate the vertices into layers, corresponding to a group or level of the hierarchy. Edges then represent affiliations. Each layer is displayed on a horizontal line, the edges as line segments between the layers. Such a representation is called an *upward layered* drawing and has been extensively studied [8, 48, 126]. The most simple, yet still very common, case considers only two layers. One example of such a representation places vertices representing organs of a biological system on one line and vertices representing cell types on a second layer. This informs the observer very clearly about the classification of each object and the inter-object relationships [126]; see Figure 1.1a for an example. Such a drawing is called a *2-layer* drawing, which has received thorough study, especially in the context of beyond planarity [15, 43, 49].

While for small graphs the representation of the full graph might fit on a regular computer screen and drawings of medium sized graphs might be printed as a large format poster, in practice users often only have a partial view of the graph. In regards to a 2-layer drawing, where the layers correspond to horizontal lines, the height of the representation is constant, thus not leading to immediate issues. However, the width of the screen is limiting what can be seen of the graph at once, assuming there is a fixed (minimum) distance in between vertices. For instance, this problem becomes apparent in a recent state-of-the-art visualization tool for anatomical structures [34]; see Figure 1.1b. While this problem seems inevitable at first, for large enough graphs, the user might not need to see the full graph all at once. However it is save to assume that the user is interested in the associations, i.e. the edges of the graph. Thus two natural questions come up: Does every edge fit completely on one screen/page (we denote this as the x-Distance), i.e. there exists a section such that we can see both incident vertices at once? Secondly, for a vertex of one of the layers, can we see all of its neighbors and the vertex itself at once (we denote this as the Window Width), i.e. can we see the vertex with its full closest context? Ideally ,we want to answer the questions with a Yes even for the smallest screen sizes. Thus, the problem studied in Part I is to construct a drawing such that the Window Width (or x-Distance) is minimal.

Upward Trees. Many hierarchical structures however contain more than just two levels. Thus automatic drawing of graphs with a more complex hierarchy has also been extensively studied. One prominent state-of-the-art method doing so is the Sugiyama framework [126], which represents hierarchical graphs in multi-layer drawings, usually while minimizing the number of crossings. Unfortunately, for general graphs, this minimization is NP-hard, even when considering only two layers, where the vertices of one layer are fixed [50].

However, many hierarchical graphs encountered in real-world applications are actually rooted trees. For instance consider phylogenetic trees [77, 103]; see Figure 1.2 for an example. These are trees representing the evolutionary history of species, showing common ancestors, where the root of a tree corresponds to a singular ancestor of a set of species. In recent years, the discipline of phylogenetic comparative biology thrived, which uses a vast array of methods to draw conclusions from comparison between multiple phylogenetic trees [25, 110, 115]. Therefore, instead of general hierarchical graphs, we study the problem of drawing multiple layered trees (and some planar graphs). While it is trivially possible to draw the trees crossing free side by side, for a direct comparison it is often more effective to display them intertwined. Specifically we consider the leaves to be sorted, for instance, ordered alphabetically to ensure efficient lookup.
While this allows the visualization to convey more encoded information, it often prevents a crossing-free drawing. Thus, the problem studied in Part II is to minimize the number of crossings, while satisfying a required leaf order, encoding additional information in the resulting visualization.

Ortho-Radial. Unsurprisingly, Graph Drawing and maps share a close and long history [66, 108, 112]. In Graph Drawing, map applications motivate incorporating geospatial information [4, 60, 121]. Map applications, on the other hand, can utilize Graph Drawing methods to generate certain types of maps automatically instead of manually [100]. With the widespread use of mobile devices, both disciplines are challenged to provide more dynamic and interactive methods.

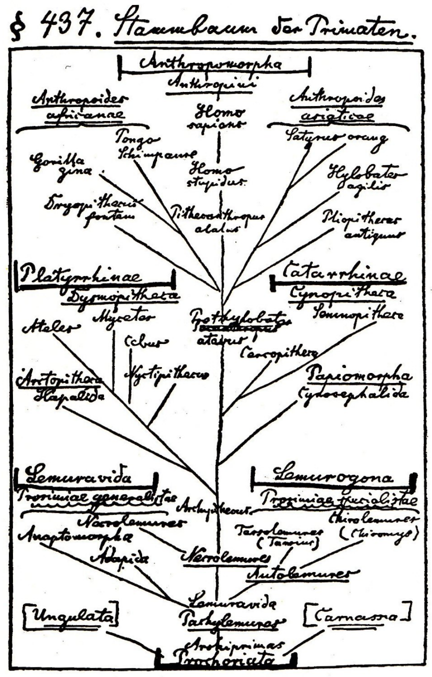
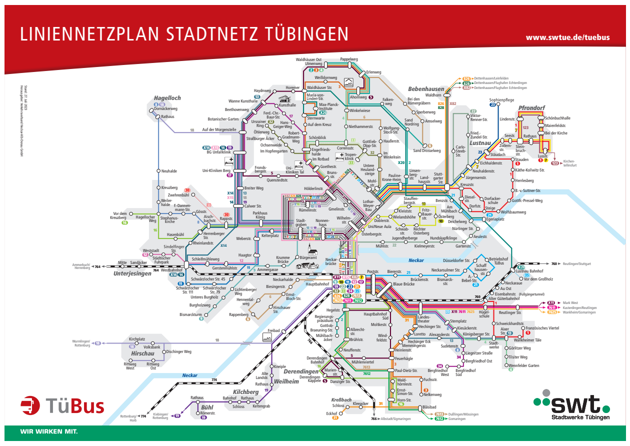


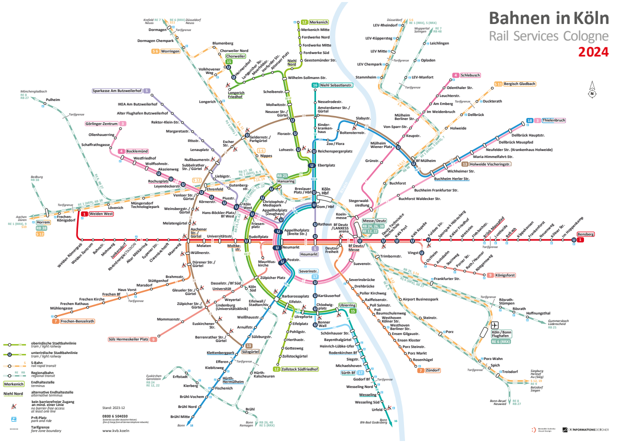
Figure 1.2: Ernst Haeckels "Stammbaum der Primaten" from the 19th century, showing the biological tree of life [68].

A very common visualization model at the intersection of both fields are metro maps; see Figure 1.3a for an example. Metro maps are widely used to schematically display public transportation systems by operators worldwide [30, 90, 91] and there are also applications in hypergraph and set visualization [58, 80]. Usually metro maps visualize the network by showing the metro lines as a color-coded path joining all stations a line traverses. However, the placement of the stations only schematically correspond to their real-world placement, maintaining orientations but relaxing correct distances in favour of a unified and simple visualization. Often the lines only consist of horizontal, vertical or diagonal (at angle of 45°) segments, which is called an *octolinear* drawing [41, 85, 127]. Another recently popular variant, called the *ortho-radial* drawing, displays the metro map in a radial fashion, such that the lines either follow concentric circles or lines emanating from the center point [6, 70, 97]. One benefit of this drawing model is that the radial representation naturally induces a focus point at the center point. So far, this has been used to emphasize some city layouts; see Figure 1.3b for an example. Here, we propose to use this style dynamically, i.e. allowing for the center point to move, for instance following the position of a user. Thus, in Part III we introduce several strategies extending the ortho-radial model to be suitable for real-time updating geospatial data.

Overview of the thesis. The remainder of this thesis is structured as follows. First we provide basic definitions and fundamental tools in Chapter 2.



(a) A metro map visualization of the public transportation system of Tübingen, taken from [65].



(b) An ortho-radial metro map visualization of the public transportation system of Cologne, taken from [39].

Figure 1.3

In Part I we formally introduce the Window Width problem and provide some theoretical advances in Chapter 3. Specifically we provide polynomial time algorithms for the minimization of the Window Width, if the bottom layer is fixed. We also provide a polynomial time algorithm for the sum of Window Widths, if the bottom layer is fixed. If the top layer is fixed we prove that the Window Width problem is NP-hard and provide a 2-approximation algorithm. Further, polynomial time algorithms for minimizing the x-Distance and the sum of the x-Distances are given.
In Chapter 4, we perform some experiments, applying the implemented algorithms on randomly generated bipartite graphs. We discuss the results and observe that, depending on the graph densities and partition ratios, there are significant improvements over a random vertex assignment.

In Part II we consider the simultaneous embedding of multiple trees with fixed leaf order. We study the problem of minimizing the number of crossings while preserving the given order. In Chapter 6 we study the case where the number of trees is restricted to two, in which we can prove that the problem is polynomial time solvable, providing a dynamic programming algorithm. In Chapter 6 we take the number of trees, k, as additional input parameter, proving that the problem is fixed parameter linear in k, if the trees are at most of height two and in XP if the trees are at most of height three. The results are achieved by reducing the problem to a shortest path problem on a k-dimensional weighted cube graph and grid graph, respectively. Additionally some generalizations of the results are provided in both Chapters 6 and 6.

In Part III we propose multiple strategies to construct an ortho-radial drawing suitable for dynamic updating. In Chapter 8 we formally define morphing of ortho-radial drawings and introduce strategies naturally supporting morphs. Additionally, we introduce the octo-radial model, which extends the ortho-radial model in a similar way as the octilinear model extends the orthogonal model. We then experimentally evaluate the strategies with respect to well-established quality metrics. To this end, the implemented strategies are applied to randomly generated data, as well as to some real-world benchmarks. In Chapter 8 a hybrid visualization model is proposed, utilizing the introduced methods.

Finally, we briefly discuss some practical application of network visualization techniques in Chapter 9 to solve five concrete challenges and conclude the thesis in Chapter 10 by summarizing the main results and presenting some open problems.

Chapter 2

Preliminaries

This section will provide the definitions and algorithmic basics necessary for the understanding of the thesis. If not specified otherwise, these definitions are based on Chapters 1 and 2 of [47].

2.1 Graph Theoretic Foundations

A *graph* $G = (V, E)$ is a pair of sets. Specifically V is the set of *vertices* of the graph G, where $E \subset [V]^2$ denotes the set of *edges* of G, thus every edge is a set of two vertices. We will write an edge $e \in E$ as (v, w), where v and w are vertices, even though e is formally a set. For better readability vertices might also be denoted as *nodes* in some parts of this thesis. For easier readability we might refer to a vertex v which is part of the vertex set $v \in V$ as $v \in G$ as well, likewise we might write $e \in G$ synonymous to $e \in E$ for an edge e. Figure 2.1 gives an example of a graph.

The *size* of G corresponds to the cardinality of its vertex set and is written as $n := |V|$. Likewise m denotes the cardinality of the edge set $m := |E|$ we will exclusively consider graphs of finite n (and thus of finite m as well) in this work.

A vertex v is *incident* to an edge e, when v is an element of e, i.e. $v \in e$. Symmetrically e is called incident to v. If two vertices v and w are incident to

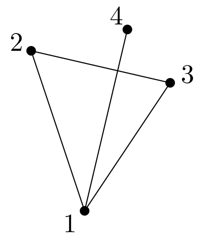


Figure 2.1: A pictorial representation of a graph $G = (V, E)$ with $V = \{1, 2, 3, 4\}$ and $E = \{(1, 2), (2, 3), (1, 3), (1, 4)\}$. Vertices are represented as dots and edges as lines connecting adjacent vertices.

the same edge $e = (v, w)$, then v is *adjacent* to w and vice versa. In such a case v and w are denoted as *neighbors*. We will refer to v and w as the *end vertices* of e. If two edges e_1 and e_2 are incident to the same vertex v, then e_1 and e_2 *join* in v.

The subset of V which includes all vertices that are neighbors to a vertex v is denoted as the *neighborhood* of v and written as $N(v)$.

The *degree* $d(v)$ of a vertex v is the number of incident edges to v, which is equivalent to the cardinality of $N(v)$.

Generally we will consider *undirected graphs*, which adhere to the given definitions. Particularly, any edge e is a set of two vertices. Thus edges are symmetrical, i.e. if v is adjacent to w then w is also adjacent to v. However in some cases we will instead consider *directed graphs* (or *digraphs*) where an edge is not a set of two vertices, but an identifier. Two functions source: $E \to V$ and target: $E \to V$ assign a *source* and a *target* vertex to every edge. Let $\text{source}(e) = v$ and $\text{target}(e) = w$, then we call e to be *directed* from v to w. We will use the notation $e = (v, w)$ for the edge. Note that in this case e corresponds to an ordered pair of vertices. We call e *incoming* to w and *outgoing* of v. Any vertex without any incoming edges is called a *source*, while any vertex without any outgoing edges is called a *sink*.

Further we will consider *simple graphs* if not stated otherwise. A directed graph is simple if it has no *self-loops* and no *multi-edges*. We call an edge e a selfloop if its source and target vertices are identical, $source(e) = target(e)$. We call at least two edges e_1 and e_2 a multi-edge, if $e_1 = (v, w) = e_2$ holds, informally, there must not be multiple copies of an identical edge. Whenever considering directed graphs we will assume they are simple graphs unless stated otherwise. Due to the definition of undirected graphs, these are naturally always simple.

In this work, a graph is considered to be undirected unless stated otherwise.

There are some special cases of graphs which come with their own notation. We will introduce the most important ones, which are relevant for this paper. If all vertices of a graph are pairwise adjacent, then the graph is a *complete graph*, which is denoted as K_n where n is the number of vertices. A K_3 is called a *triangle*.

We denote the *union* of two graphs G and G' as $G \cup G' = (V \cup V', E \cup E')$ and the intersection of G and G' as $G \cap G' = (V \cap V', E \cap E')$. If $G \cap G' = \emptyset$ then we call G and G' *disjoint*. If we consider two graphs $G = (V, E)$ and $G' = (V', E')$ where $V' \subseteq V$ and $E' \subseteq E$ then G' is called a *subgraph* of G and G a *supergraph* of G'. We write this as $G' \subseteq G$. Naturally any graph G is a subgraph of the complete graph K_n, where n is the size of G.

For a vertex subset $V' \subseteq V$, we say that V' *induces* a subgraph G' of G where $G' = (V', E')$. The induced edge set E' is the edge subset of E where both end vertices of e are elements of V'.

A *path* P is a sequence of vertices $v_1, \ldots, v_{|P|}$ such that v_i and v_{i+1} are adjacent for any $0 < i < |P|$ and all vertices of the path are distinct. We denote this as a path *from* v_1 *to* $v_{|P|}$, where we refer to v_1 as the *start vertex* and to $v_{|P|}$ as the *end vertex* of the path. The *length* of a path is $|P| - 1$. Note that a path of length 0 is possible and consists of only a single vertex. We might also refer to a path as a sequence of edges, $e_1, \ldots, e_{|P|-1}$ where two consecutive edges e_i and e_{i+1} for $0 < i < |P| - 1$ join in vertex v_{i+1}. For undirected graphs it must hold that an edge (v_i, v_{i+1}) exists for subsequent vertices v_i and v_{i+1} of the path.

If for a path P v_1 and $v_{|P|}$ are incident, then the vertex sequence $v_1, \ldots, v_{|P|}, v_1$ is called a *cycle*.

A graph G is called *acyclic* if there exists no cycle within graph G.

The *distance* $dist(v, w)$ from vertex v to vertex w is the length of the shortest path from v to w. If there exists no such path, then $dist(v, w) := \infty$.

A vertex subset V' is *connected* if for any pair of vertices $v, w \in V'$ there exists a path from v to w. If V' is not connected we call it *disconnected*. If the same holds for a vertex subset V' of a directed graph, then V' is *strongly connected*. If V itself is connected, we call the graph G *connected* as well. The *connected components* of a graph are a set of vertex subsets $V_1, \ldots, V_c$ such that each vertex subset V_i for $1 \leq 1 \leq c$ is connected and c is minimum. A graph is *k-connected* if the graph remains connected when at most $k - 1$ vertices are removed from the vertex set and the incident edges are removed from the edge set. Therefore 1-connected graph is the same as a connected graph. A 2-connected graph is called *biconnected*. A 3-connected graph is called *triconnected*.

For some applications we will consider *weighted graphs*, which are graphs with an additional weight function $w : E \to \mathbb{N}_0$, which assigns every vertex a weight. The *weighted length* of a path P of a weighted graph is the sum of the weights of all edges included in the path, i.e. $\sum_{e \in P} w(e)$. The *weighted distance* of two vertices v and w is analogously defined as the minimum weighted length of all paths from v to w.

2.2 Graph Classes

In the following we will define some *graph classes*, which are common and special families of graphs.

If a connected graph is acyclic, we call it a *tree*, see Figure 2.2 for an example. If a graph is disconnected, but all graphs induced by the connected components are trees, then we call the graph a *forest* as it consists of disjoint trees. Any vertex of a tree of degree one is called a *leaf*. For any tree it holds, that it contains exactly $n - 1$ edges. In this work, we will also encounter *rooted trees* which are trees for which one vertex is fixed and the *root* of the tree denoted as r. The root will not be considered to be a leaf even when it has degree one.

A graph G is *bipartite*, if V can be partitioned into two vertex sets A and B, such that every edge of G is incident to both one vertex of A and one vertex of B. This means, that no two vertices of the same part can be adjacent. Figure 2.3 displays an example of a bipartite graph. Often, when defining a bipartite graph, the parts A and B are already given as part of the graph definition, as in $G = (A \cup B, E)$.

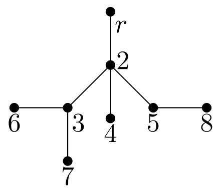


Figure 2.2: A rooted tree consisting of 8 vertices and 7 edges. The vertices 4, 6, 7 and 8 are leafs. Root r is not a leaf.

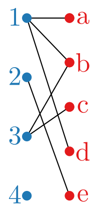


Figure 2.3: A bipartite graph consisting of vertex parts $A = \{1, 2, 3, 4\}$ and $B = \{a, b, c, d, e\}$.

2.3 Graph Drawing

For a given graph G, a *drawing* Γ of G is a function mapping every vertex $v \in V$ to a point $\Gamma(v) \in \mathbb{R}^2$ in the Euclidian plane and each edge $e \in E, e = (u, v)$ to an open Jordan curve $\Gamma(e)$ with endpoints $\Gamma(u)$ and $\Gamma(v)$. We will say, that a graph G *admits* a drawing Γ and that a curve $\Gamma(e)$ *represents* the edge e and *places* vertex v at $\Gamma(v)$.

If for two edges e_1 and e_2 the interior of their representations $\Gamma(e_1)$ and $\Gamma(e_2)$ have at least one point in common, then e_1 and e_2 *cross* in Γ. The common point is called an *intersection*.

A drawing is a *simple drawing*, if the following three conditions are met:

(i) Edges incident to one common vertex do not cross in Γ.

(ii) Two crossing edges have at most one common intersection, i.e. they do not cross multiple times.

(iii) Every edge is represented by a simple curve, i.e. it does not intersect itself.

Since every graph admits a simple drawing we will assume that all considered drawings are simple drawings. Figure 2.4 shows the forbidden configurations of simple graphs.

We call Γ a *planar drawing* if no pair of edges of G cross in Γ. A graph G is called a *planar graph* if it admits a planar drawing. A planar drawing subdivides the plane $\mathbb{R}^2$ into topologically connected regions. We refer to these regions as *faces*. Note that there exists exactly one unbounded face, which is called the *outer face*. We define a face f as a cyclic order of the vertices along the boundary of the face $(v_1, \ldots, v_{\ell(f)})$ where $\ell(f)$ is chosen such that the order of vertices is repeating after $v_{\ell(f)})$ and $\ell(f)$ is minimum. We call $\ell(f)$ the *length* of face f. Figure 2.5 provides an example.

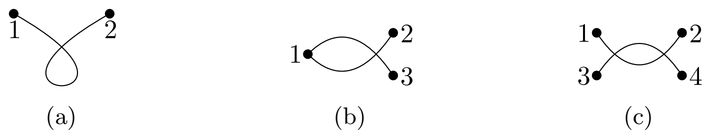


Figure 2.4: (a), (b) and (c) show three forbidden configurations for simple drawings. In (a) edge (1,2) intersects itself. In (b) edges $(1,2)$ and $(1,3)$, cross, although both are incident to vertex 1. In (c) edges $(1,2)$ and $(3,4)$ intersect multiple times.

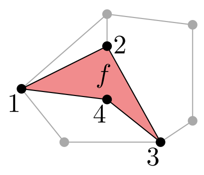


Figure 2.5: The face $f = (1, 2, 3, 4)$ of length 4 is highlighted in red.

For any planar graph the *Euler characteristic* holds which says that $n-m+f = 2$ for every planar embedding on the plane, where f denotes the number of faces of the embedding.

Two planar drawings Γ_1 and Γ_2 are *topologically equivalent* if there exists a bijection ϕ from the set of faces F_1 of Γ_1 to the set of faces F_2 of Γ_2 such that for every face $f_1 \in F_1$ the cyclic order of f_1 and $\phi(f_1)$ are identical. A class of topologically equivalent drawings of G are called a *planar embedding*, or *embedding* for short, of G denoted as ε. Observe that for all drawings of embedding ε the cyclic order of edges incident to a vertex v are the same. We refer to the tuple (G, ε) as a *plane graph*. We might also call a graph G a *plane graph* at times, implicitly assuming that a planar embedding ε exists.

A graph is an *outerplanar graph* if it admits an *outerplanar drawing*, which is a planar drawing where all vertices are on the boundary of the outer face. Analogously as with planar embeddings an *outerplanar embedding* is a non-empty class of topologically equivalent outerplanar drawings and an *outerplane graph* is the tuple of a graph and an outerplanar embedding.

A graph is called *maximal* for a given graph class, if the graph is element of the graph class however no edge can be added to the edge set E such that the resulting graph G' is also element of the graph class. For instance a graph is a *maximal planar graph* if it is a planar graph and cannot be augmented with any edge such that the resulting G' is a planar graph. A plane graph is called *triangulated* if its embedding is a triangulation of the Euclidian space, i.e. all faces of the graph are of length 3, thus all faces induce a triangle. Observe that every maximal planar graph is triangulated and that every non-maximal planar graph can be augmented with additional edges to be triangulated. A planar graph with a given embedding with fixed outer face is called *internally triangulated* if all faces which are not the outer face are of length 3.

The *dual graph* $G^* = (V^*, E^*)$ of an embedded planar graph G has a vertex for every face of G, i.e. the set of faces of G denoted as F is equivalent to the set

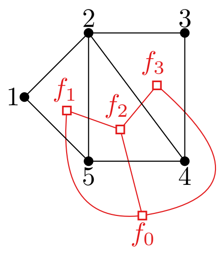


Figure 2.6: Graph $G = (\{1, 2, 3, 4, 5\}, \{(1, 2), (2, 3), (3, 4), (4, 5), (5, 1), (2, 5), (2, 4)\})$ with dual graph $G^* = (\{f_0, f_1, f_2, f_3\}, \{(f_1, f_2), (f_2, f_3), (f_3, f_0), (f_0, f_1), (f_0, f_2)\})$ shown in red. Face f_0 is the outer face.

of vertices of G^*, $V^* = F$. For two faces f_1 and f_2, the edge $e = (f_1, f_2)$ is an element of E^* exactly if f_1 and f_2 share an edge in G. Observe that the dual graph of the dual graph is identical to the original graph. Figure 2.6 provides an example for a dual graph. The *weak dual graph* of an embedded planar graph G is the subgraph of G^* induced by $F \setminus \{f_0\}$ where f_0 denotes the outer face of the embedding of G. For an outerplanar graph G, the weak dual graph is a tree.

2.4 Graph Drawing Models

We will introduce some common graph drawing models which restrict the representation of edges and the placement of vertices. However we will first define some useful tools which are essential for many of these drawing models.

Consider a graph G and two arbitrary vertices s and t of G. An *st-ordering* π of a graph is a permutation $\pi = (v_1, \ldots, v_n)$ of the vertices of G such that $v_1 = s, v_n = t$ and for every vertex $v_j, 1 < j < n$ there exist vertices v_i and v_k which are neighbors of v_j and $1 \leq i < j < k \leq n$, i.e. every v_j is sorted in between two neighbors. Such an st-ordering exists for every 2-connected graph [89].

An *st-planar graph* is a bipolar orientation of a plane graph (G, ε) that is, G is converted to a directed graph such that s and t are two vertices on the outer face of ε, the edges of G are directed such that G is acyclic and s is the only source while t is the only sink of G [41].

Let $\pi = (v_1, \ldots, v_n)$ be a permutation of the vertices of G of a maximal planar graph G. For a given embedding ε of G with (v_1, v_2, v_n) as the fixed outer face, let $V_k := \{v_1, \ldots v_k\}$ for $1 \leq k \leq n$. Let C_k denote the outer face of $\varepsilon[V_k]$ which is the embedding induced by the vertices V_k. Then π is a *canonical ordering* if and only if for $2 \leq k \leq n$ [38]:

(i) G_k is internally triangulated and biconnected.

(ii) The neighborhood of v_k which is part of G_{k-1} appears consecutively on C_{k-1}.

(iii) if $k < n$, v_k has at least one neighbor v_l such that $k < l$.

A canonical order can be computed in linear time, even for graphs which are not maximal planar, but planar and triconnected [82].

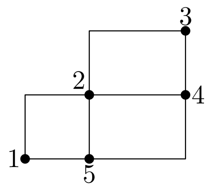


Figure 2.7: A planar orthogonal drawing of graph
$G = (\{1, 2, 3, 4, 5\}, \{(1, 2), (2, 3), (3, 4), (4, 5), (5, 1), (2, 5), (2, 4)\})$.

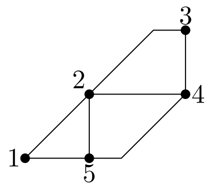


Figure 2.8: A planar octilinear drawing of graph
$G = (\{1, 2, 3, 4, 5\}, \{(1, 2), (2, 3), (3, 4), (4, 5), (5, 1), (2, 5), (2, 4)\})$.

With these tools in mind, we define some common graph drawing models.

If Γ represents all edges as straight-line segments, then Γ is called a *straight-line drawing*. If Γ places all edges on points with integer coordinates, then Γ is called a *grid drawing*. If Γ represents all edges as sequences of straight-line segments, then we call Γ a *polyline drawing* and the points where two subsequent straight-line segments join *bends*.

An *orthogonal drawing* Γ of a graph G represents every edge $e \in E$ as a sequence of horizontal and vertical line segments $\Gamma(e)$. Thus any orthogonal drawing is a polyline drawing. Additionally for a vertex v we define four *ports* called north, south, west and east. Let edge e be incident to v, then $\Gamma(e)$ has a straight line segment with v as one endpoint, depending on v being the bottom (top, left, right) endpoint of the segment, we say that e *uses* the north (south, west, east) port of v. Figure 2.7 provides an example of an orthogonal drawing. For a more detailed overview on orthogonal graphs see [41, 85, 127]. One main benefit of orthogonal drawings is the optimal *angular resolution*, which denotes the minimal angle between incident edges of a vertex.

An *octilinear drawing* Γ of a graph G represents every edge $e \in E$ as a sequence of horizontal, vertical and diagonal (at 4°) segments. Informally octilinear drawings add diagonals to the orthogonal drawing model. Accordingly for every vertex v of an octilinear drawing there exist 8 *ports* called north, north-east, east, south-east, south, south-west, west and north-west. The north port corresponds to straight line segments on the grid line eminating from c traversing through v, where v is the centermost point of the segment. All other ports follow in clockwise order. Figure 2.8 provides an example for an octilinear drawing. Octilinear drawings are common for the display of metro-maps and map schematization [75, 125].

An *ortho-radial drawing* Γ of a graph G can be considered as a translation of orthogonal drawings in polar coordinates. The graph is drawn on a ortho-radial grid consisting of M concentric circles and N spokes, which are half-lines with the

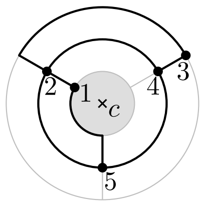


Figure 2.9: A planar ortho-radial drawing of graph $G = (\{1, 2, 3, 4, 5\}, \{(1, 2), (2, 3), (3, 4), (4, 5), (5, 1), (2, 5), (2, 4)\})$. The grid with $M = 3$ and $N = 3$ is shown in gray and the inner face is highlighted. c is usually not shown in the final drawing.

center point c as an endpoint. Edges of G are represented as sequences of *circular segments* which are segments of a concentric circle and *spoke segments* which are segments of a spoke. The junction of two subsequent segments is called a *bend* or *bend point*. Contrasting the notion of the outer face, the face which includes the center point is called the *inner face*. Ortho-radial drawings are a more recent model than the orthogonal drawings, however they received increasing attention [6, 70, 97]. Figure 2.9 provides an example for an ortho-radial drawing.

An *upward drawing* Γ of a directed graph G places every vertex such that for every $e = (v, w) \in E$ it holds that the y-coordinate of $\Gamma(v)$ is strictly lower than the y-coordinate of $\Gamma(w)$. We say that v is placed *below* w. Note that a directed graph admits a upward drawing if and only if it is acyclic.

A *layered drawing* (also sometimes known as a *hierarchical drawing*) Γ of a directed graph G is an upward drawing of G where vertices are placed on a sequence of horizontal lines $(L_1, \ldots, L_\ell)$ which are ordered increasingly by their y-coordinate. These lines are called *layers* and every vertex is placed on one layer. Every edge is represented by a curve in between the layers of its two endpoints. The maximal index ℓ denotes the *height* of the drawing. Often the height of a layered drawing should be minimal [8]. Trivially every acyclic graph admits a layered drawing on at most $\ell = n$ layers. If a given directed graph does not admit an upward drawing, it has at least one cycle. Thus often the first step of the algorithm removes edges to make the graph acyclic. However minimizing the number of edges to be removed is a NP-hard problem called the *feedback arc set* problem [17, 86]. Minimizing the number of crossings in a layered drawing is also NP-hard, however many heuristic algorithms exist [22, 126, 127].

One well studied subclass of the layered drawings restricts the number of layers to 2. These drawings are called *2-layer drawings* [15, 43, 119]. Observe that for 2-layer drawing with fixed assignment of vertices to the layers, edge crossings depend only on the order of the vertices within each layer. This is only true since the edges are limited to be represented between the two layers and we consider a simple drawing. Considering a connected graph in a two layer drawing, the assignment of the vertices to the layers is unique, apart from interchanging the two layers. This is due to the restriction that edges cannot exist between two vertices of the same layer. If the graph is not connected the same holds for every connected component and for most applications, the components can be solved separately. It has been shown that even for this restricted case minimizing

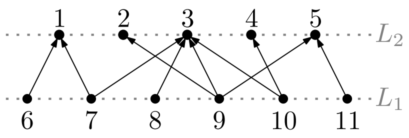


Figure 2.10: A 2-layer drawing preserving upwardness. Since the direction of the edges is implied by the drawing style, displaying the arrows is optional.

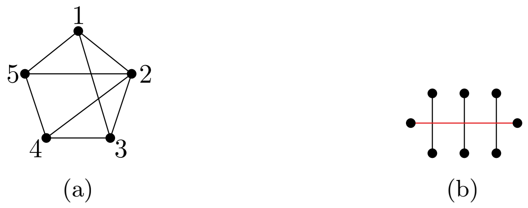


Figure 2.11: (a) A 2-planar drawing, where the edge $(1, 3)$ has the maximum of two intersections. (b) An instance showing the forbidden configuration of 2-planar drawings, where the red edge has 3 crossings.

the number of crossings is NP-hard [61]. Moreover minimizing the number of crossings remains NP-hard when the vertices of one layer are given in a fixed order [50]. Figure 2.10 provides an example for a 2-layer drawing.

2.5 Graph Drawing Beyond Planarity

In recent years a research direction has shown increasing popularity which studies the *beyond planar* graphs. Beyond planar graphs are graphs where the idea of planarity is extended by allowing some specific crossings. This is done by defining forbidden crossing configurations which are not allowed in a beyond planar drawing. This line of research was motivated by experimental findings which suggested that while crossings affect the readability of a drawing, the specific properties of crossings have a higher impact than just the quantity [76, 99]. As a consequence a large number of beyond planar graph classes were introduced and studied in recent years. Here we will show some common classes, for a more extensive overview we refer to [46].

A *k-planar drawing* Γ of a graph G is a drawing for which each edge is crossed by at most k other edges, where $k \geq 1$. Figure 2.11 provides an example for a 2-planar drawing and its forbidden configuration. k-planar graphs were already considered in the 1960s [111].

A *h-quasiplanar drawing* Γ of a graph G is a drawing for which no $h \geq 1$ pairwise crossing edges exist. Note that 2-quasiplanar drawings are the same as 1-planar drawings. Figure 2.12 provides an example for a 3-quasiplanar drawing and its forbidden configuration.

A *fan-planar drawing* Γ of a graph G is a drawing for which no three edges e, e_1, e_2 exist such that e crosses both e_1 and e_2 but e_1 and e_2 share no common endpoint [84]. Also, when e_1 and e_2 share a common endpoint v, then e must not cross e_1 and e_2 with different orientation regarding to v. Figure 2.13 provides an example for a 2-planar drawing and the forbidden configurations.

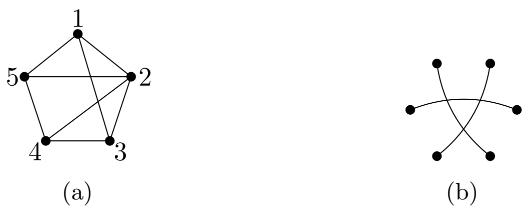


Figure 2.12: (a) A 3-quasiplanar drawing, while both $(2, 5)$ and $(2, 4)$ cross $(1, 3)$, $(2, 5)$ and $(2, 4)$ do not cross. (b) An instance showing the forbidden configuration of 3-quasiplanar drawings, with 3 pairwise crossing edges.

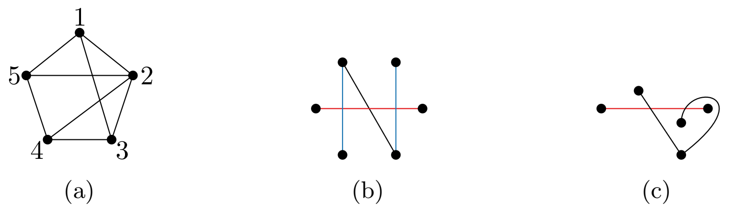


Figure 2.13: (a) A fanplanar drawing, both edges crossing $(1, 3)$ share a common vertex 2. (b) A forbidden configuration of fanplanar drawings, where both blue edges are crossed by the red edge, but they are not incident to a common vertex. (c) Another forbidden configuration of fanplanar drawings, the red edge is crossed by two edges with a common vertex, but with different orientations.

A *RAC drawing* Γ of a graph G is a straight drawing for which all edge crossings occur at a right angle [44].

2.6 Further Frequently Used Definitions

A *matching* of a graph $G = (V, E)$ of size $n := |V|$ is a subset of edges $M \subseteq E$ such that every vertex of V is incident to at most one edge of M. A vertex which is incident to one edge of M is called *matched*. If the vertex is not incident to any edge of M it is called *unmatched*. The *size* of a matching corresponds to the cardinality of M. If for a matching M, there exists no edge $e \in E$ for which $M \cup \{e\}$ is a matching, then M is a *maximal matching*. If for a matching M there exists no other matching M' of G of larger size, then M is a *maximum matching*. A matching M is a *perfect matching* if every vertex of V is matched, i.e. the size of M is exactly $\frac{n}{2}$. A *weighted matching* is a matching M on a weighted graph, the *weighted size* of M is the sum of the weights of all edges of M, i.e. $\sum_{e \in M} w(e)$. A *weighted maximum matching* is likewise a weighted matching M such that the weighted size of no other weighted matching M' is larger. Note that a weighted maximum matching is not necessarily a maximum matching. A *maximum weight maximum matching* is a weighted matching M with maximum weighted size out of the set of maximum matchings of G. Likewise a *minimum weight maximum matching* is a weighted matching M with minimum weighted size out of the set of maximum matchings of G.

A *Delaunay triangulation* of a point set P in the plane is a triangulation of P such that for every triangle (p_1, p_2, p_3) of the triangulation the circle defined by the three points p_1, p_2 and p_3 is empty, i.e. there is no other vertex of P in the interior of this triangle. A Delaunay triangulation can be used to compute an internally triangulated graph, when a drawing of a vertex set without any edges is provided. Further, if the convex hull of P is a triangle, the graph corresponding to the Delaunay triangulation is a maximal planar graph and fully triangulated. Delaunay triangulations maximize the minimum angle of all triangles in the triangulation over all triangulations. A Delaunay triangulation can be computed in running time $\mathcal{O}(|P| \log |P|)$ [88].

An *FPT* or *fixed parameter tractable* problem, is a problem which can be solved in time $f(k) \cdot n^{\mathcal{O}(1)}$, for a computable function f, where k is a parameter of the input independent of n. Therefore the class of FPT algorithms allows for a more fine-grained classification of NP-hard problems, describing problems which become polynomial time solvable, when k is bound by a constant. One special subclass of FPT-problems are *FPL* or *fixed parameter linear* problems, for which an algorithm exists, such that the running time is linear, if k is constant, i.e. the problems can be solved in time $f(k) \cdot n$. For example the `SAT` or `Vertex Cover` problems are in FPL. Lastly, *XP* is the class of problem, for which an algorithm of running time $n^{f(k)}$ exists, where f is a computable function, clearly XP is a superclass of FPT and FPL. Still, when k is constant, a polynomial time algorithm exists for XP problems.

Part I

Window Width

Chapter 3

Theoretical Results

In this chapter of the thesis we will introduce the notion of window width for bipartite graphs and study different variants of the window width minimization problem, where the vertices of one of the layers of the given 2-layer drawing have to be replaced such that the maximum distance spanned by any vertex of the top layer and its neighborhood is below a given threshold. We observe that this problem can be solved efficiently when the bottom layer is fixed and the top vertices can be freely placed on integer coordinates and NP-complete when the top layer is fixed and the bottom vertices can be freely placed. However we will provide a 2-approximation algorithm for this case. Additionally we consider the closely related notion of x-Distance, where the maximum difference in x-coordinate between any adjacent vertices has to be minimized, as well as a summed variation of both the window width and the x-Distance problems, where the accumulated Window Widths or x-Distances have to be minimized and provide polynomial time algorithms solving both.

Some results of this chapter also appeared in [9][1] which received the Best Student Paper Award.

3.1 Window Width Minimization with Flexible Top Layer

Acyclic directed graphs are common for many real-world networks such as organizational charts [72], function trees [116] or parse trees [28], as these graphs describe a hierarchy between elements. Such graphs are sometimes also called *hierarchical graphs*. State of the art algorithms use a layer by layer approach, thus considering only two subsequent layers at a time, to construct a representation of hierarchical graphs [126]. Therefore the 2-layer case is of special importance. Such drawing models are even used for directed graphs which are not acyclic, were a few non-upward edges are accepted.

Moreover 2-layer drawings are widely used to visualize bipartite graphs for any data set, where there are two groups or communities and their inter-group

[1]All collaborators contributed to equal parts to the results of this paper.

relationship is studied. A state of the art visualization directly motivating this research is the ASCT+B REPORTER, displaying anatomical structures, biomarkers and cell types in a 2-layer drawing [34]. While the visualization succeeds at compactly conveying medical knowledge, it is cumbersome to find all biomarkers belonging to a cell type, as a considerable amount of scrolling is necessary. Many similar visualizations exist, such as tanglegrams for phylogenetic trees [122][37][24][54] where the leaf layers of two separate phylogenetic trees are opposed and edges between the layers display matching taxa, thus the two leaf layers are visualized using a 2-layer drawing.

An essential task in exploration of most visualizations is to identify the neighborhood of a vertex. In the example stated above this corresponds to identifying all biomarkers belonging to a cell type, or all cell types belonging to an organ. Many interactive graph visualizations even highlight the incident edges and according neighbors when selecting (or hovering over) a vertex. Clearly, for easy readability, the highlighted section should be visible in full, thus the vertex and its neighborhood must fit into the display window of the application. This motivates the definition of a new optimization criteria, which will be studied in this chapter.

Let $G = (A \cup B, E)$ be a bipartite graph which is directed such that any vertex is directed from its incident vertex in A to its incident vertex in B. Let n_A denote the number of vertices of A, n_B the number of vertices of B, $n := n_A + n_B$ the total number of vertices and $m := |E|$ the total number of edges. Let Γ be a 2-layer drawing of G, observe that all vertices of A are on layer L_2 and all vertices of B on layer L_1. As there are only two layers we will also refer to L_1 as the *bottom layer* and to L_2 as the *top layer*. For a connected bipartite graph this orientation and subsequent assignment to layers is unique apart from interchanging A and B. For vertex v of a 2-layer drawing Γ the x-coordinate of v is denoted as $x(\Gamma(v))$. As all vertices are assigned to one layer, the y-coordinate of all vertices is clear. Therefore we might refer to the x-coordinate simply as the *position* of v in Γ. We will only consider positions on integer coordinates. The *x-Distance* denoted as $xs_\Gamma(v, w)$ between two vertices v, w in a 2-layer drawing Γ is the difference of x-coordinate of v and w, i.e. $xs_\Gamma(v, w) := |x(\Gamma(v)) - x(\Gamma(w))|$. If the placement of some vertices is provided by a function p (as it mostly will be in this chapter) we instead denote the x-coordinate of v as $p(v)$. The *Window Width* of vertex $v \in A$ in respect of a given drawing Γ is the maximum x-Distance between any two vertices of $\{v\} \cup N(v)$, i.e. between vertices of the neighborhood of v including v. We denote the *Window Width of a vertex* v as $ww(v) := max_{u,v \in \{v\} \cup N_v} xs_\Gamma(u, v)$. We denote the *Window Width of a drawing* Γ as ww_Γ and define it as the maximum Window Width of any vertex of A, $ww_\Gamma := max_{v \in A} ww(v)$. Figure 3.1 shows an example illustrating the x-Distance, the Window Width of a vertex and the Window Width of a drawing. Finally the *Window Width of a graph* is denoted as $ww_G := min_{\Gamma \in \varepsilon} ww_\Gamma$ where ε denotes the set of 2-layer drawings on integer coordinates of bipartite graph G. Note that we will restrict the set of drawings ε in the following further by taking the positions of all vertices of one vertex set A or B as part of the input. Such a restricted Window Width will be denoted as $ww_A(G)$ or $ww_B(G)$ respectively. Only the vertices of the B or A respectively can be replaced.

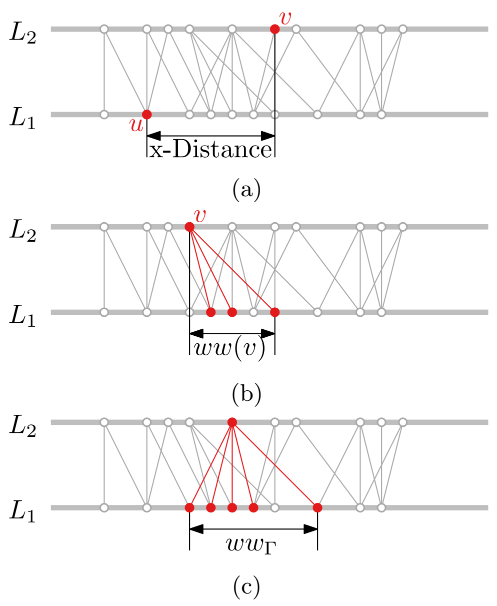


Figure 3.1: (a) The x-Distance between two vertices u and v. (b) The Window Width of vertex $v \in A$. (c) The Window Width of Γ, which is the maximum Window Width for any vertex of A.

Given a partial 2-layer drawing, where the x-coordinates of the vertices of B on the bottom layer L_1 are fixed and the vertices of A can be freely assigned, we provide an efficient algorithm to construct a 2-layer drawing of G with minimum Window Width.

Theorem 3.1. *Given a bipartite graph $G = (A \cup B, E)$ as well as a vertex placement $p : B \to \mathbb{Z}$. A 2-layer drawing Γ of G with $x(\Gamma(v)) = p_B(v) \forall v \in B$ and minimum Window Width ww_B can be constructed in $\mathcal{O}(n_A \log(ww_B) + m)$ time.*

Proof. As an outline, the algorithm will first reduce the problem to an equivalent but (generally) smaller problem, then calculate a lower bound for the Window Width and set it as temporary Window Width. We will then try to construct a drawing satisfying the temporary Window Width with a sweep line algorithm by placing vertices of A within intervals which satisfy the Window Width. If this fails, the intervals are enhanced and the temporary Window Width increased. The algorithm can continue after increasing the Window Width without having to start again. A final pass will compute the correct placements for the vertices of A satisfying the final temporary Window Width which equates to the optimal Window Width.

Reducing the Problem. We first observe that the instance can be reduced to the *critical part* of G. For any vertex $v \in A$ only the leftmost neighbor denoted as $\ell(v)$ at position $p(\ell(v))$ and the rightmost neighbor denoted as $r(v)$ at position $p(r(v))$ are potentially relevant for Window Width of v. For any 2-layer drawing of G complying with placement p, the Window Width of the critical part of G is

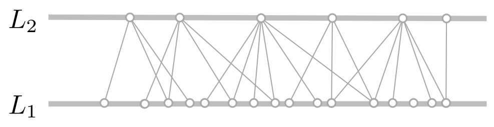


Figure 3.2: The running example for this section.

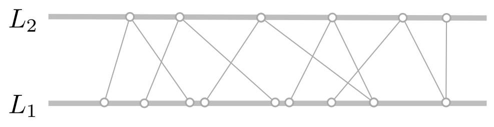


Figure 3.3: The equivalent critical part of the running example. Note that every vertex of A has at most degree 2.

equivalent to the Window Width of G. To show this, assume first that v will be placed at an x-coordinate which is inside the interval $[p(\ell(v)), p(r(v))]$. Then $ww(v) = p(r(v)) - p(\ell(v))$, thus any other neighbors can be omitted. Secondly assume that v will be placed at x-coordinate x outside of $[p(\ell(v)), p(r(v))]$. W.l.o.g. assume that $x > p(r(v))$, then $ww(v) = x - p(\ell(v))$ thus again, all other neighbors are inconsequential. Therefore we can remove all intermediate edges in $\mathcal{O}(m)$ time resulting in a critical part with $\mathcal{O}(n_A)$ vertices. Note that $\ell(v) = r(v)$ is possible. Vertices of B without incident edges are irrelevant for the placement of A, similarly any vertices of A without incident edges are neglected, as those can be placed on any free x-coordinate. In the following, we assume that the graph G has already been reduced to its critical part. Figure 3.2 provides a running example. In the following, only the critical part of the example will be considered, shown in Figure 3.3.

Computing a Lower Bound. The algorithm does begin with a *temporary Window Width* denoted as k_{ww} which is incremented during the algorithm, whenever the algorithm fails to construct a drawing of k_{ww} until the optimal Window Width is achieved. Observe that the maximum x-Distance between $\ell(v)$ and $r(v)$, i.e. $p(r(v)) - p(\ell(v))$ for any $v \in A$, is a natural lower bound for ww_B as placing v within the interval $[p(\ell(v)), p(r(v))]$ yields exactly this Window Width and placing v outside of the interval yields a strictly higher Window Width. Thus initially k_{ww} is set to $\max_{v \in A} p(r(v)) - p(\ell(v))$. Figure 3.4 shows the lower bound and initial k_{ww} for the running example.

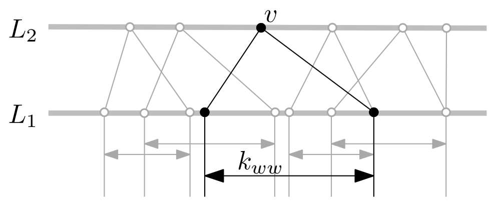


Figure 3.4: The lower bound for the Window Width (in this case $ww(v)$) defines the initial k_{ww}.

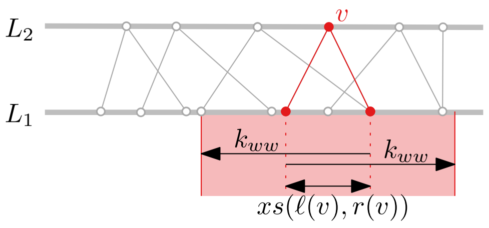


Figure 3.5: The interval $I(v)$ of vertex v, observe that it is of size $2k_{ww} - xs(\ell(v), r(v))$.

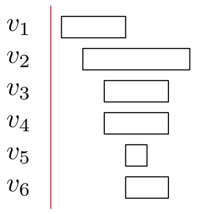


Figure 3.6: Showing the initial intervals of all vertices of A as a new (smaller) running example. Note that it suffices to observe the intervals corresponding to the vertices. Vertex placements are displayed as point within the interval at a given coordinate. Each coordinate can only be assigned once. The sweep-line shown in red is empty at initiation.

Constructing the Intervals. Let $I(v) := [p(r(v)) - k_{ww}, p(\ell(v)) + k_{ww}]$ be the *interval* of any vertex v which defines the set of positions where vertex v can be placed, such that the Window Width of vertex v is at most k_{ww}. Note that there is at least one vertex with interval size of exactly k_{ww} and all intervals have at most size $2 \cdot k_{ww}$, see Figure 3.5. Let $\min I(v)$ denote the leftmost coordinate in $I(v)$ and $\max I(v)$ the rightmost coordinate in $I(v)$. Now it is necessary and sufficient to find a unique position of every vertex of A within $I(v)$ to construct a Γ such that $ww_\Gamma = k_{ww}$. To solve this efficiently, we introduce a sweep-line algorithm which allows for k_{ww} to be increased during the sweep.

The Sweep-Line. The intervals of the vertices of A will be swept from left to right by a vertical sweep-line denoted as L. L is a data-structure maintaining the set of currently *active intervals*. The active intervals are all intervals of vertices of A which are intersected by L and whose vertex in A has not been assigned a position yet. In the data structure, these active intervals are sorted by their right endpoints, which can be achieved by a min-heap [35]. During the sweep we distinguish between three types of events, denoted as *start*, *placement* and *end* event. It is possible that multiple events occur at the same x-coordinate, in that case start events take priority, placement events are handled only when there are no start events and end events are only resolved when no other events exist on the given coordinate. Figure 3.6 provides an example, the intervals have been constructed before the sweep-line starts. In the following the event types are described in detail:

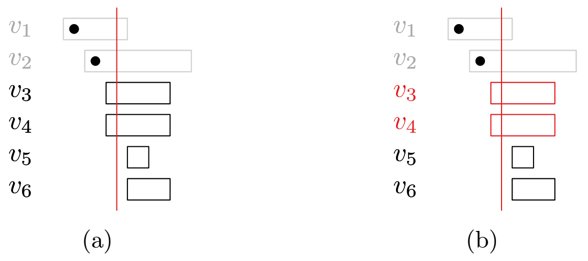


Figure 3.7: Showing the sweep-line (a) before and (b) after two start events adding $I(v_3)$ and $I(v_4)$ as active intervals.

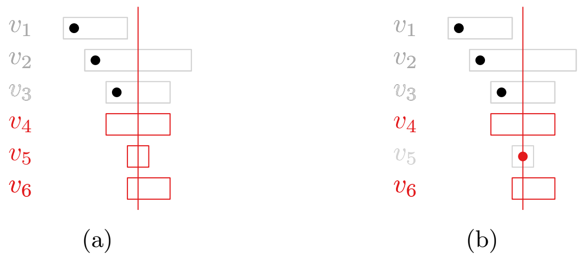


Figure 3.8: Showing the sweep-line (a) before a placement event, where $L = (I(v_5), I(v_6), I(v_4))$ as they are sorted by their endpoints and (b) after the placement event occurred, assigning v_5 to the current position and marking its interval as inactive.

Start event. A start event adds an active interval to L at, such that it can be considered by the other events.
This event type occurs for any vertex $v \in A$ at the position of the left endpoint of $I(v)$. The event inserts $I(v)$ to L. If there is no placement event at the current position, there is a new one added. Figure 3.7 shows a start event in the running example.

Placement event. A placement determines the ideal candidate for placement at the current position out of all active intervals and places it.
This event type occurs whenever the data structure of L is non-empty, i.e. there are active intervals. In this case we can assign one vertex of A to the current x-coordinate x, so $p(v) := x$ where v is the first active interval of L. $I(v)$ is then marked as *inactive* and removed from L. If L is non-empty, a placement event at $i + 1$ is added. Note that there are exactly n_A placement events in total, as any placement event assigns exactly one vertex to a x-coordinate. Figure 3.8 shows a placement event in the running example.

End event. An end event controls that each vertex is assigned within its interval. If an interval ends although its vertex has not been placed yet, the temporary Window Width k_{ww} is incremented.
This event type occurs whenever L coincides with the right endpoint x of an interval $I(v)$. To construct a drawing with Window Width of k_{ww}, the interval has to be inactive, i.e. its vertex has already been assigned to a position within the interval, so that $x_\Gamma(v) \leq x$ holds. If L is inactive we can proceed, otherwise

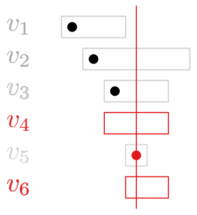


Figure 3.9: Showing an end event which does not increment k_{ww}, $I(v_5)$ is ending, but $I(v_5)$ is inactive, thus nothing has do be done.

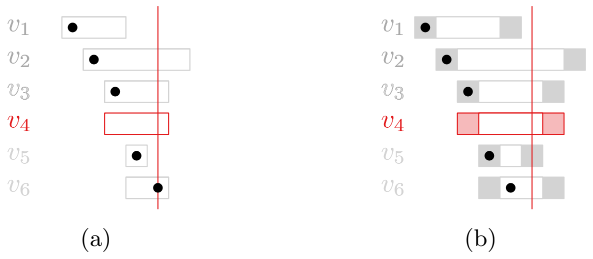


Figure 3.10: Showing the sweep-line (a) before an end event for v_4 and (b) after an end event for v_4. The temporary Window Width k_{ww} is incremented as $I(v_4)$ is still active. Observe that all intervals are increased by 1 in each direction and the assignments which already have been made are shifted by 1 to the left. Thus the current position becomes free.

we failed to find a position of v within its interval $I(v)$. Thus a drawing with the temporary Window Width cannot be realized, k_{ww} has to be incremented, which extends all intervals by one on each side. All already placed vertices of A and all start events are moved one to the left and all end events one to the right along the $x-$axis. Then all placement events are removed and a new placement event at x introduced. Note that due to this shift there might be unresolved start events at current position x. This is the case when there was a start event at $x+1$ before the shift. Therefore the next event handled is either a start or a placement event, still at position x. Since all placements already made have been shifted to the left, position x is free and can be assigned a vertex. Figure 3.9 shows an end event in the running example where the interval is inactive and Figure 3.10 an event, where the interval is active and thus k_{ww} is incremented.

Before the sweep begins, all start events and end events are placed in the event queue in $\mathcal{O}(n_A)$ time. Note that to efficiently handle the increments of k_{ww}, start events and end events are kept in separate event queues which are processed synchronously. The order of the events within the separate queues are maintained naturally when k_{ww} changes, as all start events are shifted by the same fixed amount of 1 to the left and all end events by 1 to the right. Also we observe that there are at most 2 placement events at any point during the algorithm, there can be one at current position x and one at the next position $x+1$. Thus it is best to realize the placement event queue as two booleans instead of a regular queue.

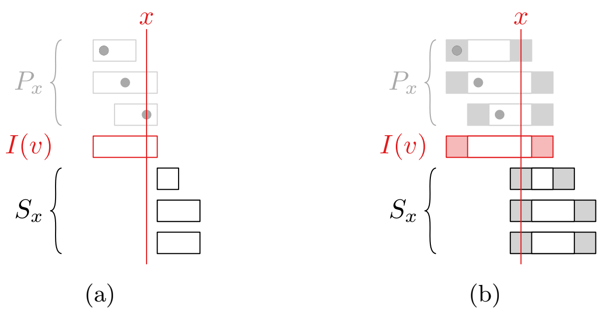


Figure 3.11: Observe that the right boundaries of the intervals of P_x are strictly left of the right boundaries of the intervals of S_x before (a) and after (b) incrementing k_{ww}.

Correctness. Clearly, by definition of the vertex intervals, when the algorithm terminates, all vertices have been assigned a unique position and the corresponding drawing Γ has Window Width $ww_\Gamma = k_{ww}$, where k_{ww} is the temporary Window Width at algorithm termination. Further we observe, that when the algorithm fails to place a vertex within its interval and as a result k_{ww} is increased, the partial solution up to this point shifted by one unit to the left is identical to the partial solution which would be obtained by restarting the algorithm with an incremented k_{ww}. Since the order of end events and start events remains the same respectively the following property holds: Assume the placement of vertex $v \in A$ within its interval $I(v) = [\ell, r]$ fails at x-coordinate x. Let $P_x \subset A$ denote the set of vertices which were already placed by the algorithm. Further let S_x be the vertices whose start event occurs at position $x+1$, therefore after increasing k_{ww} it occurs at x. Then after resolving the end event, it holds that $\max I(p) \leq \max I(s)$ for any $p \in P_x, s \in S_x$. This is true since $\max I(p) \leq r = x+1$ and $x = \min I(s) < \max I(s)$ unless $k_{ww} = 0$ which is impossible, as k_{ww} was just incremented. This proves that all vertices of P_x will be placed at positions left of $x+1$ and all vertices of S_x at positions right of $x+1$ (as position $x+1$ is assigned to v or a vertex with an equivalent interval). The order of placements is unaffected by the increment of k_{ww}. Figure 3.11 visualizes this property.

Thus it only remains to show that k_{ww} is only increased if necessary, i.e. when a drawing of Window Width k_{ww} is impossible. If a free coordinate within $I(v)$ exists, then v would have been placed at this position contradicting our assumption, as $I(v)$ was the only active interval at this coordinate. Therefore all of the coordinates of $I(v)$ were assigned to previously placed vertices of A. Let ℓ' denote the largest x-coordinate $< r$ which has not been assigned a vertex of A, i.e. ℓ' is free. Clearly $\ell' < \ell$. Let $A_v \subset A$ be the set of vertices placed within interval $I_v := [\ell'+1, r]$. It must hold that for a solution of Window Width k_{ww} all vertices of A_v must be placed within interval I_v. Assume for a contradiction that vertex $a \in A_v$ exists which can be placed outside of I_v while maintaining Window Width of k_{ww}. This is only possible, if the vertex a is placed within $I(a)$. However $\max I(a) \leq r$ since a has been assigned a coordinate previous to v by the algorithm. Also $\min I(a) > \ell'$ otherwise a would have been placed at ℓ' as it was an active interval at this position. Thus all vertices of A_v as well

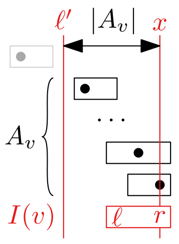


Figure 3.12: Schematic representation of the proof showing that a drawing of Window Width k_{ww} is impossible, as the interval $[\ell', r]$ is overfull.

as v must be placed within interval I_v in a drawing with Window Width k_{ww}, however there are only $|A_v|$ coordinates in I_v to place $|A_v| + 1$ vertices, which is a contradiction. Figure 3.12 illustrates this proof. □

Time complexity. Reducing G to its critical part takes $\mathcal{O}(n + m)$ time. As there are exactly one start, one end and one placement event per vertex of A plus one end event per increment of k_{ww} there are $\mathcal{O}(n_A + ww_B) = \mathcal{O}(n_A)$ events in total, since there are at most $\Theta(n_A)$ increments as ww_B has a trivial upper bound of $\max\{\max_{v\in A} p(r(v)) - p(\ell(v)), n_A\}$ (recall that $\max_{v\in A} p(r(v)) - p(\ell(v))$ is the lower bound of ww_B).
To keep track of the unresolved events, two sorted event queues for start and end events suffice, there are at most two placement event at any algorithm step. In L all active intervals have to be sorted. Observe, that there can be at most $\mathcal{O}(ww_B)$ many active intervals at any algorithm step, more precisely at most $2ww_B + 1$. We show this by contradiction. Assume that at some x-coordinate x there are $2ww_B + 2$ intervals maintained in L. Since any vertex must be placed on a unique integer coordinate, there must exist one vertex v with interval $I(v) = [\ell, r]$ which is placed earliest at x-coordinate $x + 2ww_B + 1$. Note that $\ell \le x$ as $I(v)$ is an active interval at position x. From the interval definition it follows that $p(r(v)) = \ell + ww_B \le x + ww_B$. Any interval is of maximum size, if $r(v) = \ell(v)$, therefore $p(\ell(v)) + ww_B \le p(r(v)) + ww_B \le x + 2ww_B$. Thus if v is assigned x-coordinate $x + 2ww_B + 1$, it is placed outside its interval, yielding a Window Width of $ww_B + 1$, which is a contradiction.

Thus L can be maintained by a min heap holding at most $\mathcal{O}(ww_B)$ intervals, therefore any event can be resolved in $\mathcal{O}(\log(ww_B))$ time.

We store an offset value which corresponds to the number of increments of k_{ww} from its initial value (the lower bound) until the termination of the algorithm. This offset corresponds to the shift of start and placement events to the left and of end events to the right, therefore the intervals do not have to be adjusted whenever k_{ww} increments. For all events it suffices to add (or subtract) the offset value. Still one additional pass over all vertices of A after the sweep line terminated is necessary to adjust all placement events, shifting $x(v)$ by this offset value $ww_B - \max_{v\in A} xs(r(v), \ell(v))$ to the left for every vertex $v \in A$. Thus realizing all increments of k_{ww} can be done in linear time.

3.2 Window Width Sum Minimization with Flexible Top Layer

While minimizing the maximum Window Width over all vertices can increase the readability of the drawing significantly, for some constellations we observe some undesired results. Specifically if there exists one vertex which prominently defines the maximum Window Width, the algorithm provided in the previous section may yield unnecessarily long edges for all other vertices. We remark that this is due to the greedy nature of the sweep-line algorithm, which, informally, places the vertices leftmost possible. This can also be observed in the experimental results in Chapter 4. Thus, instead of focusing on the maximum Window Width of every single vertex a natural variant of the problem is derived by minimizing the average Window Width of all vertices. Note that this is equivalent to minimizing the sum of Window Widths of all vertices of A. In the following, we show that this variant of the problem is also solvable in polynomial time, when the vertices of B are given with a fixed placement.

Theorem 3.2. *Given a bipartite graph $G = (A \cup B, E)$ as well as a vertex placement $p : B \to \mathbb{Z}$. A 2-layer drawing Γ where the sum of Window Widths of all vertices of A is minimized while preserving the given vertex placement for B can be constructed in $O(n_A^3 + m)$ time.*

Proof. We will show this by providing a polynomial time algorithm with the stated running time. The algorithm models the instance of our problem as a bipartite matching problem. This can be efficiently solved by the well known Hungarian algorithm to find a minimum weight maximum matching [109]. Remark, that the size of the problem can still be reduced by removing unnecessary edges as in Theorem 3.1. Therefore after reducing the problem to its critical part, the primary step of the algorithm is to construct a weighted bipartite graph, denoted as $G^* = (A^*, B^*)$.

Before discussing the construction of G^*, we observe two important properties: For any vertex $v \in A$ the optimal placement of v is within the interval spanned by the x-coordinates of $\ell(v)$ and $r(v)$, as $p(r(v)) - p(\ell(v))$ is a general lower bound for $ww_\Gamma(v)$ for any Γ respecting p. Further for any positions outside of this interval, the Window Width of v increases linearly with the distance to the interval. As the coordinates assigned by p must not be linear, we argue in the following, that the number of positions we have to consider is still limited. We do so by defining an extended interval for every vertex v which has size $\geq n_A$, thus at least one position within the interval must be free, as there are only $n_A - 1$ other vertices of A. Further all positions outside of the extended interval yield a higher Window Width for v. Specifically we define the extended interval as $[p(\ell(v)) - \frac{n_A}{2}, p(r(v)) + \frac{n_A}{2}]$ if $p(r(v)) - p(\ell(v)) < n_A$. As the Window Width of v increases with distance to the $[p(\ell(v)), p(r(v))]$ all positions outside of the extended interval clearly yield a higher Window Width for v, also, as $p(\ell(v)) \leq p(r(v))$ the extended interval is clearly of size at least n_A. If $p(r(v)) - p(\ell(v)) \geq n_A$ it suffices to define the extended interval as any subinterval of $[p(\ell(v)), p(r(v))]$ of size exactly n_A. For instance let the extended interval be $[p(\ell(v)), p(\ell(v)) + n_A]$. Clearly the interval is of size n_A, further all coordinates within the interval yield the same, minimal, Window Width

$p(r(v)) - p(\ell(v))$ for v.
Thus we have shown that it suffices to consider $\mathcal{O}(n_A)$ positions per vertex. Now for the construction of G^*.

Constructing G^*. For each vertex v we construct the extended interval of relevant placements, as defined above, each of size n_A. Then, for each position x, which is element of at least one such interval, we introduce a node denoted as v_x to node set A^*. Additionally for each vertex v of A, we introduce a node v^* to B^*, v^* can be considered a copy of v. The edge set corresponds to introducing an edge connecting every vertex of A with all positions within its extended interval. The edge weights are then defined such that they correspond to the Window Width resulting from placing vertex v at the incident position. Thus formally edge set E^* contains an edge $e = (v_x, v^*)$ exactly if $x \in [p(\ell(v)) - \frac{n_A}{2}, p(r(v)) + \frac{n_A}{2}]$, in case $p(r(v)) - p(\ell(v)) < n_A$ and exactly if $x \in [p(\ell(v)), p(\ell(v)) + N_A$, in case $p(r(v)) - p(\ell(v)) \geq n_A$. The weight $w((v_x, v^*))$ is defined as the x-coordinate distance between x and the interval $[p(\ell(v)), p(r(v))]$ plus $p(r(v)) - p(\ell(v))$, that is

$$w((v_x, v^*)) = \begin{cases} p(r(v)) - x, \text{if } x < p(\ell(v)) \\ x - p(\ell(v)), \text{if } x > p(r(v)) \\ p(r(v)) - p(\ell(v)), \text{if } p(\ell(v)) \leq x \leq p(r(v)). \end{cases}$$

Observe that the constructed G^* consists of n_A nodes of B^* and at most n_A^2 nodes of A^*, as there are exactly n_A extended intervals each of size n_A, thus $\mathcal{O}(n_A^2)$ nodes in total. Note however, that when the coordinates assigned with p are linearly bounded by n_A, there are only $\mathcal{O}(n_A)$ nodes. As any vertex of B^* has a degree of n_A, there are $\mathcal{O}(n_A^2)$ edges in total.

Now as G^* is constructed, any known algorithm can be used to compute a minimum weight maximum matching M^* of G^*. Since $|B^*| = n_A$ and any node of B^* has degree n_A, clearly any maximum matching of G^* is of size n_A, i.e. in the resulting matching M^* all nodes of B^* are matched. If an edge $e = (v_p, v^*)$ is an element of M^*, we consider vertex v of A to be placed at position p. Observe, that the weight of e corresponds exactly to the Window Width of v when placing v at position p. Thus M^* corresponds to a drawing Γ, optimally positioning all vertices of A, minimizing the sum of Window Widths $\sum_{v \in A} ww_\Gamma(v)$.

Correctness. First we verify that M^* corresponds to a valid coordinate assignment of the vertices of A. As M^* is a matching, no node $v^* \in B^*$ has two incident edges in M^*, thus no vertex of A is assigned two positions. Similarly, as no node $v_x \in A^*$ has two incident edges, no two vertices of A are assigned the same x-coordinate. As any node $v^* \in B^*$ has a degree of n_A it trivially holds that $|M^*| = n_A$, i.e. all vertices of A have been assigned.
To argue that the placement is optimal, assume for a proof by contradiction there exists a solution in which the sum of Window Widths of all vertices of A is strictly smaller than the one corresponding to the minimum maximal matching M^*. Clearly, this optimal positioning can be expressed as a matching M' of G^* with $w(M') < w(M^*)$, which is also maximal as it is of size n_A. By definition of the edge weights this implies that M^* is not minimum, which is a contradiction.

Time complexity. As in Theorem 3.1 the critical part of G can be computed in $\mathcal{O}(n_A + m)$ time. Any node of B^* has degree n_A, B^* consists of n_A nodes and A^* consists of $\mathcal{O}(n_A^2)$ nodes. There are $\mathcal{O}(n_A^2)$ edges in G^*. Computing the weight of an edge takes constant time, thus in total $\mathcal{O}(n_A^2)$ time for all edges. Thus in total, the construction of G^* takes $\mathcal{O}(n_A^2)$ time. The minimum weight maximum matching of G^* can be computed by the Hungarian algorithm in time $\mathcal{O}(n_A^3)$ [109]. Therefore the total time complexity is $\mathcal{O}(n_A^3 + m)$. □

3.3 Window Width Minimization with Fixed Top Layer

Since the Window Width defined as the maximum Window Width of any vertex of vertex set A, the Window Width is not a symmetrical property. Therefore having the positions of the top vertices fixed, while placing the vertices of B is an entirely different problem. In fact unless N = NP deciding whether $ww_A(G) \leq k$ for a fixed k can not (deterministically) be done in polynomial time. We denote this as the Window Width problem.

Theorem 3.3. *Given a bipartite graph $G = (A \cup B, E)$, an integer k and a vertex placement $p : A \to \mathbb{Z}$, it is* NP*-complete to decide whether there exists a 2-layer drawing Γ of G with $x(\Gamma(v)) = p(v), \forall v \in A$, such that $ww(\Gamma) \leq k$.*

Proof. It is obvious that the problem is a member of NP. The hardness proof can be adapted from a proof by Papadimitriou, reducing from the Exact 3-SAT problem, to the Bandwidth problem [104]. In preparation, the concept of the reduction by Papadimitriou will be briefly described first. For the Bandwidth problem, a graph $G = (V, E)$ is given and the vertices have to be assigned on a line with unique integer positions $f : V \to \mathbb{Z}$, such that the distance spanned by any edge is bound by a given k, called the *Bandwidth*. Formally $|f(v) - f(w)| \leq k, \forall (v, w) \in E$.
The Exact 3-SAT problem is a NP-complete variation of the SAT problem, where a Boolean formula φ is given, where φ consists of n variables and m clauses, where any clause consists of exactly 3 different literals. The problem is to decide, whether φ is satisfiable.

Papadimitriou's reduction The core concept in the reduction from Exact 3-SAT to Bandwidth is the construction of an instance graph G for which the Bandwidth problem is true for a fixed k exactly if there exists a solution to the Boolean formula of the Exact 3-SAT problem. To do so, G is constructed such that it consists of blocks, which are subgraphs $\mathcal{H}$, each containing a *literal-vertex* for all literals of φ, i.e. both ℓ_x and $\ell_{\neg x}$ for each variable x. Further $\mathcal{H}$ includes two vertices denoted by M and M'. The subgraphs are constructed in such a way, that exactly n literal-vertices can be placed left of M and M' while the other n literal-vertices must be placed right of M and M'. The set of literal-vertices on the left are denoted as P and the one on the right as Q. If a solution of Bandwidth k exists, the set of literal-vertices included in P corresponds to the set of satisfied literals, while the literal-vertices included in Q correspond to the unsatisfied literals.
To construct G $n + m$ copies of the subgraph $\mathcal{H}$ are combined. These are adapted such that a 'consistency' of literals is assured, by that we mean that any literal-vertex which is in P in any one copy of $\mathcal{H}$ should be in P in all other

copies of $\mathcal{H}$. This automatically ensures that the same holds for literal-vertices in Q. This can be achieved by using the bandwidth constraint. By adding edges to G it becomes impossible to have a vertex both in P and Q for different copies of $\mathcal{H}$ without violating the bandwidth constraint. Further, the first n copies of $\mathcal{H}$ are adjusted to ensure that every variable x of φ has exactly one literal ℓ_x or $\neg\ell_x$ in P while the other literal is in Q. Finally the clauses of φ are implemented by the last m copies of $\mathcal{H}$ where a vertex is introduced which is incident to all literals of a given clause and ensures that at most two of its vertices can be in Q, so that φ is fulfilled, else the bandwidth constraint is violated.

Our reduction Observe that the WINDOW WIDTH problem differs greatly from the BANDWIDTH problem, as the BANDWIDTH problem consists of a single layer instead of two and all edges are considered individually instead of neighborhoods as in the Window Width problem. Further in the WINDOW WIDTH all vertices of the top layer have fixed positions, while in the BANDWIDTH problem all vertices can be freely assigned x-coordinates. However we are able to utilize the main concept of Papadimitrious algorithm to reduce the EXACT 3-SAT problem to the WINDOW WIDTH problem. Since the vertices of A are given as an input, it is easier to fix vertices of vertex set B to specific positions, simplifying the construction of some gadgets. However these constraints make the flexibility of this model, which is necessary to show NP-hardness, less apparent.

In our reduction, we construct a graph $G = (A \cup B, E)$ and a x-coordinate assignment $p : A \to \mathbb{Z}$, such that $ww_A(G) = 6n+3$ if and only if Boolean formula φ is satisfiable. Recall that n denotes the number of variables and m the number of clauses of φ. We introduce a number of gadgets, which will be subgraphs of G, achieving similar behavior as their counterparts in Papadimitriou's reduction. Thus, when appropriate, we maintain the nomenclature of Papadimitriou's proof.

To construct sequences in which the literals are separated into P and Q, i.e. into satisfied and unsatisfied literals, we introduce *$\mathcal{H}$-gadgets*, where the *block-gadget* *B_2-block* corresponds to M and M' in Papadimitriou's construction.
Similarly the 'consistency' of literals of our construction is ensured by introducing vertices in A in a *propagation gadget*. Analogously to Papadimitriou's construction, the first n copies of the $\mathcal{H}$-gadget realize *variable-gadgets* ensuring that either the positive or the negated literal is satisfied, while the last m copies realize the clauses via *clause-gadgets*.

In the following, we will describe the construction of these gadgets in detail. Each gadget introduces one or two vertices to A with appropriate positions p. Block-gadgets and $\mathcal{H} - gadgets$, the 'building blocks' of our construction, additionally introduce vertices to B. Without loss of generality we assume that there are at least five variables, i.e. $n \geq 5$.

Block-gadget. A Block-gadget introduces a number of β vertices to B which are by construction fixed to a sequence of x-coordinates $i, \ldots, i + \beta - 1$, therefore making it impossible for any other vertex of B to be placed on any of these coordinates. We call these vertices the *block-vertices*, as they informally block positions which can no longer be assigned to any flexible vertices. To achieve this property, two vertices denoted as a_ℓ and a_r, are introduced to A with $p(a_\ell) := i - (k - \beta + 1)$ and $p(a_r) := i + k$. Both vertices are incident to all

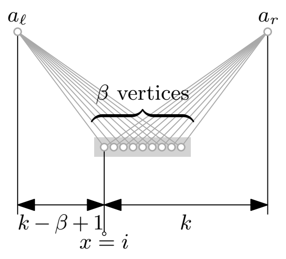


Figure 3.13: A block-gadget of size β.

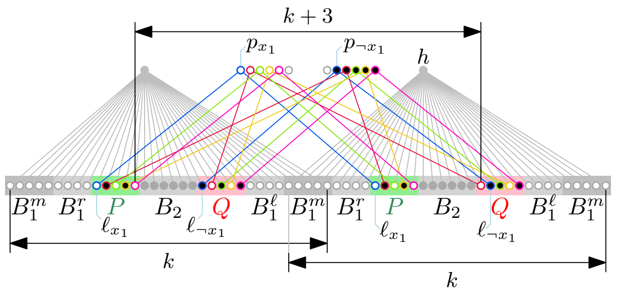


Figure 3.14: A $\mathcal{H}$-gadget and propagation-gadget.

block-vertices, see Figure 3.13 for a depiction of the block-gadget construction. Clearly a Window Width of k can only be admitted if any block-vertex is placed at a position with x-Distance at most k to both a_ℓ and a_r. This is only fulfilled exactly when every block-vertex is placed within the interval $[i, i+\beta-1]$ in B, which only consists of exactly β positions. Note that within this interval, the block-vertices can be freely reordered.

In our construction of G, two types of block-gadgets are used. A B_1-block consisting of $\beta_1 = 2n+3$ block-vertices and a B_2-block consisting of $\beta_2 = n+1$ block-vertices. Further, the block-vertices of any B_1-block are partitioned into three vertex sets, B_1^ℓ, B_1^m and B_1^r. B_1^m consists of n block-vertices and will be placed in the middle, due to constraints of the $\mathcal{H}$-block. B_1^ℓ consists of $\lfloor \frac{n+3}{2} \rfloor$ vertices, while B_1^r consists of the remaining $\lceil \frac{n+3}{2} \rceil$ block-vertices. See Figure 3.14 for a depiction.
B_2-blocks serve to separate the *P-blocks* from the *Q-blocks*. Left of any B_2-block (except for the first), there is a P-block and right of any B_2-block (except for the last), there is a Q-block. Both the P- and Q-blocks span over n x-coordinates, so that they provide enough positions for n literal-vertices each. B_1-blocks then separate the copies of these sequences of P, B_2, Q-blocks. Thus any B_1-block has a Q-block on its left and a P block on its right. conversely to the B_2 blocks. In total there are $n+m+1$ B_2-blocks and $n+m$ B_1-blocks which alternate. Note that combining one full sequence of these blocks, i.e. a B_1-,P-,B_2- and a Q-block (in exactly this order) results in a total amount of $5n+4$ vertices of vertex set B. Thus this gives us the size $\psi := 5n+4$ of one *cycle of blocks* which defines the coordinates of all blocks of our construction as in Table 3.1.

Note that for the i-th B_1-block, this results in a placement of vertex a_ℓ at $p(a_\ell) := \psi \cdot (i-2) + n + 4$, which corresponds to the position above the $n+4$-th

Table 3.1: Block Coordinates

Block Type	**First x-coordinate**	**Last x-coordinate**
B_1-block	$\psi \cdot (i-1)+1$	$\psi \cdot (i-1)+2n+3$
P-block	$\psi \cdot (i-1)+2n+4$	$\psi \cdot (i-1)+3n+3$
B_2-block	$\psi \cdot (i-1)+3n+4$	$\psi \cdot (i-1)+4n+4$
Q-block	$\psi \cdot (i-1)+4n+5$	$\psi \cdot (i-1)+5n+4$

The x-coordinate of the i-th B_1-, P-, B_2-, and Q-blocks enumerated from left to right with $i \geq 1$.

vertex of the previous B_1-block, while a_r is placed at $p(a_r) := \psi \cdot i + n$, which corresponds to the n-th vertex of the next B_1-block, in a left-to-right manner respectively. Analogously for the i-th B_2-block, this results in a placement of vertex a_ℓ at $p(a_\ell) := \psi \cdot (i-2)+3n+5$, which corresponds to the position above the second vertex of the previous B_2-block, while a_r is placed at $p(a_r) := \psi \cdot i+4n+3$, which corresponds to the n-th vertex of the next B_2-block, in a left-to-right manner respectively.

$\mathcal{H}$-gadget. These gadgets introduce the *literal vertices* to B. For any variable x_i of Boolean formula φ, two vertices are introduced, corresponding to literals ℓ_{x_i} and $\ell_{\neg x_i}$ respectively. Hence, there are $2n$ vertices. Further, each $\mathcal{H}$-gadget includes one B_2-block denoted as b, which separates the P-block preceding b from the Q-block succeeding b. Any two consecutive $\mathcal{H}$-gadgets are then separated by a B_1-block, limiting the placement of all literal-vertices to exactly the before-mentioned P- and Q-blocks. See Figure 3.14 for a depiction of two consecutive $\mathcal{H}$-gadgets. An $\mathcal{H}$-gadget can be constructed as follows: For any B_2-block b we introduce an $\mathcal{H}$-gadget H which consists of one vertex $h \in A$ and $2n$ literal-vertices of B. h is incident to all vertices of b and the introduced literal-vertices. Further, it is incident to the B_1^m- and B_1^r vertices of the preceding B_1-block and the B_1^ℓ- and B_1^m vertices of the succeeding B_1-block. B_1^ℓ- and B_1^r denote all vertices of a B_1-block, excluding B_1^m separated into two equal parts, if n is odd. If n is even, we define B_1^ℓ to include one vertex more than B_1^r. Note that two consecutive $\mathcal{H}$-gadgets share n vertices incident to their respective h vertex, namely B_1^m. For the $\mathcal{H}$-gadget associated with the i-th B_2-block b, $p(h) := \psi \cdot (i-1)+3n+4$, i.e. h is placed above the leftmost vertex of B_2-block b. Since the leftmost vertex of B_1^m preceding b and the rightmost vertex of B_1^m succeeding b are at distance exactly k, all vertices incident to h must be placed in between these two B_1^m blocks. Otherwise the resulting Window Width is greater than k. Note that in the following construction, block-vertices receive no further incident edges, therefore Window Width k can only be obtained, if the vertices of B_1^ℓ are placed left of B_1^m and the vertices of B_1^r are placed right of B_1^m for all B_1-blocks. Also note, that the literal vertices are only restricted to be in either the i-th P-block or the i-th Q-block for the i-th $\mathcal{H}$-gadget so far. There are not yet any restrictions on the partition of the vertices into P and Q or the order of these vertices within the P- and Q-blocks.

We will assume that the vertices in P-blocks correspond to satisfied literals while the vertices in Q-blocks correspond to unsatisfied literals in the following.

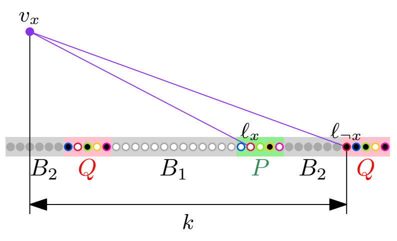


Figure 3.15: A variable-gadget.

Propagation-gadget. So far the literal vertices must be placed in the P or Q blocks of the respective copy of the $\mathcal{H}$-gadget, but they can be completely freely assigned. The propagation-gadget now aims to maintain 'consistency' of satisfied and unsatisfied literals. To achieve this, for every copy of B_1 a vertex is introduced for every literal of φ. Considering variable x_j we introduce two *propagation-vertices* p_{x_j} and $p_{\neg x_j}$. Let H_i and H_{i+1} denote two consecutive $\mathcal{H}$-gadgets. Both H_i and H_{i+1} are incident to a common vertex set B_1^m. Let b be the B_1-block of B_1^m. Then vertex p_{x_j} is connected to ℓ_{x_j} in both H_i and H_{i+1}, while vertex $p_{\neg x_j}$ is connected to $\ell_{\neg x_j}$ in both H_i and H_{i+1}. Note that b is the i-th B_1-block. Vertex p_{x_j} is placed at position $p(p_{x_i}) := \psi \cdot (i-1) + (j-1)$ and vertex $p_{\neg x_j}$ is placed at position $p(p_{\neg x_i}) := \psi \cdot (i-1) + (n+4) + j$. This ensures that all propagation-vertices assigned to negated literals are placed to the right of the a_ℓ vertex above b, while all propagation-vertices assigned to positive literals are placed to the left of the a_r vertex above b. This implies, that the leftmost propagation-vertex is placed above the rightmost literal-vertex of the Q-block preceding b, while the rightmost propagation-vertex is placed above the leftmost literal-vertex of the P-block succeeding b.
Therefore, the Window Width of these propagation-vertices is determined only by the placement of their two incident literal-vertices and is unaffected by the placement of the propagation-vertex itself. Further, we observe that the rightmost position of the P-block of H_i has distance $k-n+1$ to the rightmost position of the P-block of H_{i+1}. This allows for any reordering of the vertices of the P block between H_i and H_{i+1}. The same holds for the Q-blocks of H_i and H_{i+1}. However, we also observe that the rightmost position of the P-block of H_i has distance $k+3$ to the leftmost position of the Q-block of H_{i+1}, thus literal-vertices corresponding to the same literal cannot be part of P in H_i and part of Q in H_{i+1}. It must remain part of the P-block, otherwise the Window Width of the propagation-vertex is greater than k. Since this holds for any literal-vertex in P, all n positions existing in the P-block of H_{i+1} are occupied by a literal-vertex and thus all literals represented in a Q-block of H_i have to be represented in a Q-block in H_{i+1} as well. This achieves 'consistency' in propagation of literals from H_i to H_{i+1}.

The construction so far guarantees that each literal is either consistently satisfied (it is part of a P-block) or unsatisfied (it is part of a Q-block) for any $\mathcal{H}$-block. We are still missing the encoding of clauses and the consistency of variables, i.e. only exactly one of ℓ_{x_j} or $\ell_{\neg x_j}$ is satisfied.

Variable-gadget. This gadget ensures that for each variable x of φ at most one of its literal-vertices ℓ_x or $\ell_{\neg x}$ can be placed in a Q-block. Since there exists one

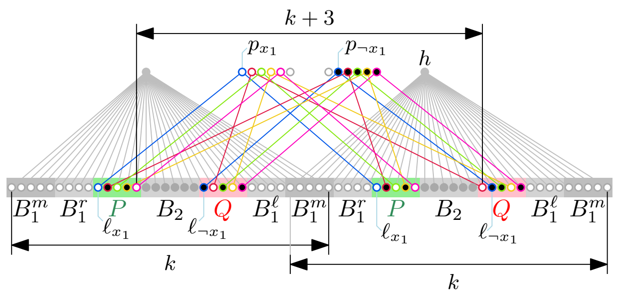
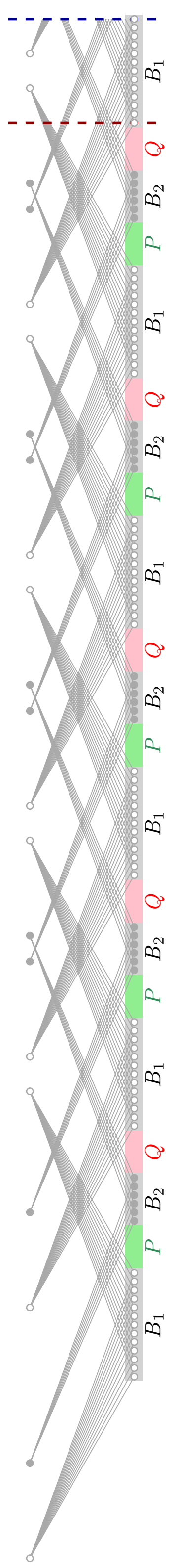


Figure 3.16: A clause-gadget.

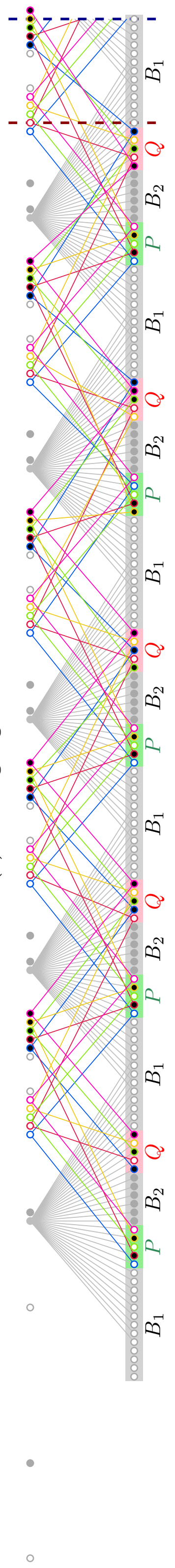
gadget for each of the n variables, there is at least one literal of each variable in the P-block. However there exist only n positions in a P-block, therefore there must also be one literal of each variable in the respective Q-block. Thus, exactly one of ℓ_x or $\ell_{\neg x}$ is in the P-block while the other literal-vertex is in the Q-block, which is then propagated by the propagation-gadget in all $\mathcal{H}$-gadgets. To achieve this, the leftmost n $\mathcal{H}$-gadgets are augmented by a variable-gadget. For each variable x, we introduce a *variable-vertex* v_x to A. The variable-vertex is incident to the literal vertices ℓ_x and $\ell_{\neg x}$ of the corresponding $\mathcal{H}$-gadget. Let the variable-gadget be associated with $\mathcal{H}$-gadget H. Vertex v_x is placed such that it is at x-Distance k to the leftmost position of the Q-block which is part of H. Specifically $p(v_x) := \psi \cdot (i-2) + 3n + 6$ if H is the i-th $\mathcal{H}$-gadget. This corresponds to the position above the third leftmost block-vertex of the B_2-block preceding H. Clearly, the rightmost neighbor of v_x can be placed at the leftmost position of the Q-block, otherwise the Window Width of v_x is greater than k. Thus, only one of v_x neighbors can be placed in the Q-block and necessarily at least one must be placed in the P-block. As argued before, this is sufficient to ensure that each variable is consistently either true of false in all $\mathcal{H}$-blocks. Figure 3.15 depicts the construction of a variable-gadget.

Clause-gadget. To encode a clause of φ, a similar construction to the variable-gadget construction can be used. Again one clause-gadget augments one $\mathcal{H}$-gadget. Since there exist m clauses in φ, the rightmost m $\mathcal{H}$-gadgets are augmented. Now consider a clause $\kappa = (\lambda_1 \vee \lambda_2 \vee \lambda_3)$ where λ_1, λ_2 and λ_3 are literals. Note that due to the definition of Exact 3-SAT these are exactly three unique literals. The clause-gadget for κ will admit a drawing of Window Width at most k if and only if at least one literal of λ_1, λ_2 and λ_3 is placed in P and, thus, satisfied. To assure this, *clause-vertex* c_κ is introduced to A in $\mathcal{H}$-gadget H. The clause-vertex c_κ is connected to the three literal vertices associated to λ_1, λ_2 and λ_3. Vertex c_κ is positioned at x-coordinate $p(c_\kappa) := \psi \cdot (i-2) + 3n + 7$, assuming that H is the i-th leftmost $\mathcal{H}$-gadget. This corresponds to the position above the fourth leftmost block-vertex of the B_2-block preceding H, see Figure 3.16. Since the vertex c_κ is at x-Distance k to the second leftmost literal vertex of Q, at most two out of λ_1, λ_2 and λ_3 can be placed within the Q-block. Thus, at least one of λ_1, λ_2 and λ_3 must be placed within the P-block of H. Therefore at least one of λ_1, λ_2 and λ_3 must be satisfied.

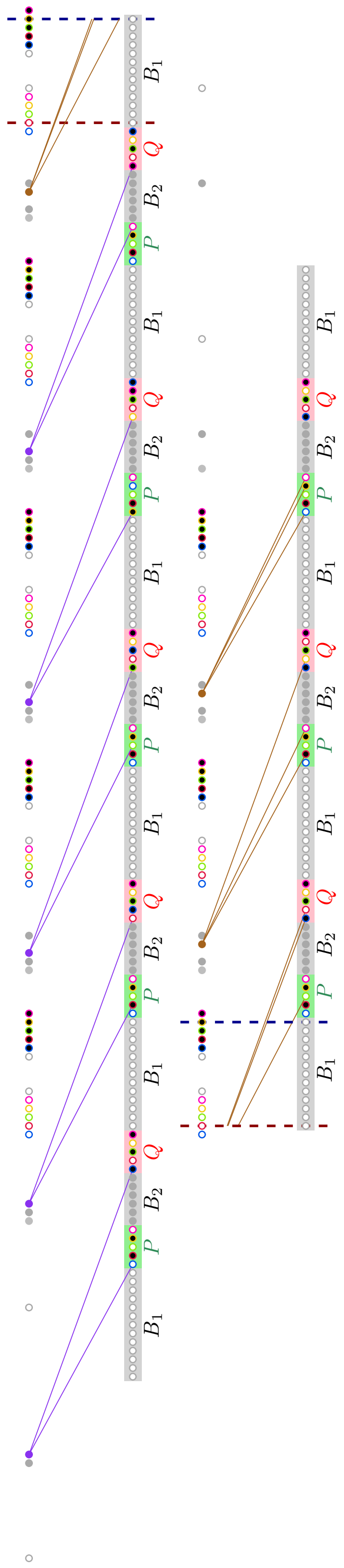
A full example of a construction for a small Exact 3-SAT instance is shown in Figure 3.17d.



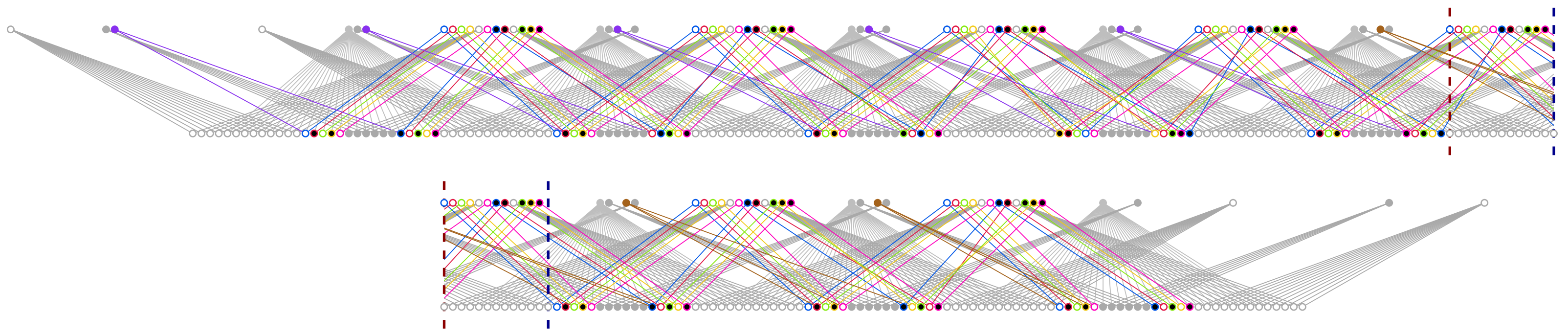
(a) Block-gadgets.



(b) $\mathcal{H}$- and propagation-gadgets.



(c) Variable- and clause-gadgets.



$\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee x_4 \vee x_5)(x_1 \vee \neg x_4 \vee x_5)$ with
$x_1 = x_3 = x_5 = \top$ and $x_2 = x_4 = \bot$

(d) Full drawing.

Figure 3.17: Example of the construction of our NP-hardness reduction for the instance $\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee x_4 \vee x_5) \wedge (x_1 \vee \neg x_4 \vee x_5)$ which is satisfied by assignment $x_1 = x_3 = x_5 = \top$ and $x_2 = x_4 = \bot$. The Literal-vertices of x_1, x_2, x_3, x_4 and x_5 are shown in colors blue, red, green, yellow and pink, respectively, and filled white if associated with the positive or black if associated with the negative literal. For clarity this figure is separated into subfigures. Subfigure (a) shows only the B-blocks. Subfigure (b) omits the edges of the B-blocks but adds the propagation- and $\mathcal{H}$-gadgets. In subfigure (c) all variable- and clause-gadgets are shown, alongside the vertices introduced in the previous subfigures. Subfigure (d) shows the full construction with all edges. Note that Subfigures (c) and (d) are split into two rows at the B_1-block indicated by the dashed lines.

To complete the construction, we have to show that the assignment of vertices to x-coordinates introduced by the gadgets is without contradiction. Clearly the placement of vertices of B is possible as discussed above. Note that only the literal-vertices and block-vertices are elements of B. As for the vertex set A, the placement of vertices is fixed, so we can verify that no pair of vertices v and $w \in A$ exists with $p(v) = p(w)$, by inspecting the placement of the gadget-vertices as in Table 3.2. These coordinates are given relative to the alternating B_1 and B_2-blocks, except for the very first a_ℓ and v_x vertices which are placed left of all other vertices. It can easily be verified that none of these positions coincide if $n \geq 5$, which we assumed at the beginning of this proof.

Table 3.2: Placement Coordinates

block	j	**vertex**
B_1-block	$[1, n-1]$	$p_{x_2}, \dots, p_{x_n}$ of propagation-gadget
	n	a_r of preceding B_1-block
	$n+4$	a_l of succeeding B_1-block
	$[n+5, 2n+3]$	$p_{\neg x_1}, \dots, p_{\neg x_{n-1}}$ of propagation-gadget
P-block	1	$p_{\neg x_n}$ of propagation-gadget
B_2-block	1	h of associated $\mathcal{H}$-gadget
	2	a_ℓ of succeeding B_2-block
	3	v_x associated with the next $\mathcal{H}$-gadget
	4	c_κ associated with the next $\mathcal{H}$-gadget
	n	a_r of preceding B_2-block
Q-block	n	$p_{\neg x_n}$ of propagation-gadget

Placements of vertices of A in reference to the block they are placed above. Note that the position of a vertex x above the j-th vertex of the i-th B_1-block is at $p(x) = \psi \cdot (i-1) + j$ while the position of x above the j-th vertex of the i-th B_2-block is at $p(x) = \psi \cdot (i-1) + 3n + 3 + j$. Further, the position of x above the j-th vertex of the i-th P-block is at $p(x) = \psi \cdot (i-1) + 2n + 3 + j$ and for the i-th Q-block $p(x) = \psi \cdot (i-1) + 4n + 4 + j$ respectively. Note that naturally none of these intervals overlap.

Equivalence of instances. To prove that φ is satisfiable if and only if there exists a drawing Γ of G where $x(\Gamma(a)) = p(a), \forall a \in A$ and $ww(\Gamma) \leq k$, we first assume that φ is satisfiable. A drawing with $ww_\Gamma = k$ can be constructed by placing all satisfied variables in the P-block and all unsatisfied variables in the Q-block of each $\mathcal{H}$-gadget. Further the literal-vertices are ordered such that the vertices incident to the variable- or clause-gadget are placed leftmost in their respective P- or Q-blocks. The resulting drawing has the desired Window Width $ww_\Gamma \leq k$. Now assume that φ is not satisfiable, however there exists a drawing Γ of G where $x(\Gamma(a)) = p(a), \forall a \in A$ and $ww_\Gamma \leq k$. By construction, each variable of φ must be either consistently true or false, i.e. within P or Q and for each clause at least one of its literals must be true. Thus, a truth assignment for φ can be obtained by observing any P-block of Γ. If a variable is part of the P-block as a positive literal, then the variable is $\top$, otherwise the variable must be $\bot$. By construction of G this assignment satisfies φ which contradicts the assumption. This concludes the proof.

Polynomial time of reduction. Observe that each $\mathcal{H}$-gadget and propagation-gadget consists of $\mathcal{O}(n)$ vertices, while the variable- and clause-gadgets consist of only one additional vertex. Since there are $m+n$ many $\mathcal{H}-$gadgets overall the graph consists of $\mathcal{O}((n+m)\cdot n)$ vertices and can be constructed also in time $\mathcal{O}((n+m)\cdot n)$ which is polynomial. □

Since the Window Width decision problem is NP-complete, the next natural step is to investigate in approximation algorithms. Indeed it turns out, that the result of Theorem 3.5, which we will receive in the next section, can be used to obtain a 2-approximation algorithm solving the optimization variant of this problem.

Theorem 3.4. *Given a bipartite graph $G = (A \cup B, E)$ as well as a vertex placement $p : A \to \mathbb{Z}$. A 2-layer drawing Γ of G with Window Width $ww_\Gamma \leq 2 \cdot ww_A(G)$ and $x(\Gamma(v)) = p(v), \forall v \in A$, can be constructed in $\mathcal{O}(n_B \log(ww_A(G)) + m)$ time.*

Proof. We obtain the result by applying the algorithm of Theorem 3.5, which minimizes the maximum x-Distance of all pairs of adjacent vertices, where the positions of vertices of A are fixed. The algorithm has a running time of $\mathcal{O}(n_B \log(k^\star) + m)$ where $k^\star$ is the minimum maximum x-Distance between any adjacent vertices. Note that we interchanged vertex sets A and B before applying the algorithm. Let k denote the Window Width ww_Γ of the drawing Γ obtained by the algorithm.
In the following, we show that $k \leq 2ww_A(G)$. First, as shown in Theorem 3.5 the x-Distance of the longest edge in Γ is bound by $k^\star$. Clearly it holds that $k \leq 2k^\star$. Equality occurs only if a vertex $v \in A$ is placed exactly in the middle between its leftmost and rightmost neighbor, having distance exactly x-Distance $k^\star$ to both. If $k > 2k^\star$, then at least one of the neighbors must have x-Distance $> k^\star$ which is a contradiction.
On the other hand, if we consider the optimal 2-layer drawing of G $\Gamma^\star$, where $ww_{\Gamma^\star} = ww_A(G)$, necessarily the longest edge in $\Gamma^\star$ spans a x-Distance of $\leq ww_A(G)$ otherwise the vertex of A incident to this edge would have a Window Width strictly larger than $ww_A(G)$, contradicting the optimality of $\Gamma^\star$. Thus, $k^\star \leq ww_A(G)$ must hold, otherwise $k^\star$ would not be optimal. Combining this with the first argument, we obtain $k \leq 2k^\star \leq 2ww_A(G)$, proving the 2-approximation. Also, by the same argument, we have shown that the time complexity of the algorithm $\mathcal{O}(n_B \log(k^\star) + m)$ is equivalent to $\mathcal{O}(n_B \log(ww_A(G)) + m)$. Note that the drawing obtained in Γ is a corresponding solution with Window Width $ww_\Gamma \leq 2ww_A(G)$. □

3.4 x-Distance Minimization with One Flexible Layer

We continue with considering another natural variant of the optimization problem. Where in the previous sections the Window Width problem is motivated by the exploration of the neighborhood of a vertex, now we consider the case where only a single edge is explored and/or highlighted. Again, naturally this edge should fit on one screen for good readability. Analogously to the Window Width problem, we first consider the longest of all edges, in respect to their x-Distance, as this restricts the overall resolution or screen size, necessary. Thus we are interested in minimizing the maximum distance of x-coordinates between any adjacent vertices by assigning vertices to integer coordinates on their layer. Note that this problem is closely related to the well studied *maximum edge-length* property [93][117], however we are still considering 2-layer bipartite graphs only. Note that we will simplify the length-metric, considering only the length in regard of the x-coordinate difference between two vertices, while for the original maximum edge-length problem, the Euclidean distance between the two layers is significant. However both metrics correlate (although non-linearly), thus the result in Theorem 3.5 holds for the maximum edge-length as well. As with the Window Width, let xs_Γ denote the x-Distance of drawing Γ, which is the maximum x-Distance of any edge. Figure 3.18 shows the x-Distance of the running example. $xs(G)$ denotes the minimal x-Distance of any 2-layer drawing of G with integer coordinates.

First, we consider the case where the positions of vertices of one layer are fixed, while the vertices of the other layer can be assigned arbitrarily. We denote this as the x-Distance $xs_A(G)$ or $xs_B(G)$, depending on whether A or B is fixed. Observe that in contrast to the Window Width problem, this problem is symmetrical, i.e. A and B can be interchanged for an equivalent problem. This is due to the x-Distance being dependent on the individual edges. However, we obtain a similar result to Theorem 3.1.

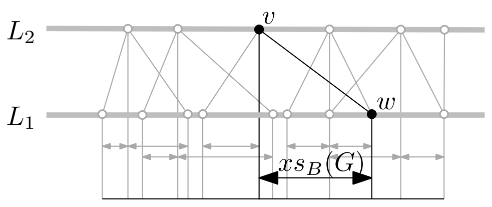


Figure 3.18: Calculating the maximum x-Distance xs_Γ of the provided 2-layer drawing.

Theorem 3.5. *Given a bipartite graph $G = (A \cup B, E)$ as well as a vertex placement $p\colon B \to \mathbb{Z}$. A 2-layer drawing Γ of G of minimum maximum x-Distance $k^\star$ between any pair of adjacent vertices, preserving the given vertex placement p, can be constructed in $\mathcal{O}(n_A \log(k^\star) + m)$.*

Proof. To prove this we provide an $O(n_A \log n_A + m)$ time algorithm, solving the problem. To this end we adjust the algorithm described in the prove of

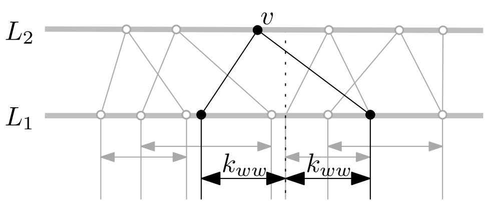


Figure 3.19: Showing the lower bound for the maximum x-Distance $xs_B(G)$. Note that the vertices of A are not yet assigned a position and therefore placed randomly. The dashed black line marks the optimal placement of v, at the mean of $p(\ell(v))$ and $p(r(v))$.

Theorem 3.1, such that the x-Distance instead of the Window Width is optimized. As in the proof of Theorem 3.1, the first step is to identify the critical part of G. Although we consider the maximum of all x-distances, it suffices to consider only the edge of vertex $v \in A$ to its leftmost and rightmost neighbors, i.e. the vertices with maximum and minimum x-coordinate respectively, as any neighbors in between yield a smaller x-Distance to v. Thus the critical part consists of $O(n_A)$ vertices and edges.

We propose an algorithm, which finds an optimal placement of all vertices $v \in A$ such that the maximum x-Distance between any pair of adjacent vertices in the critical part is minimized. We denote this maximum x-Distance as $k^\star$. Note that the maximum x-Distance of the critical part is equivalent to the maximum x-Distance of G, thus $k^\star$ is also the optimal maximum x-Distance of G.

As in the proof of Theorem 3.1, for each vertex v we define an *interval* $I(v)$, which consists of all x-coordinates where v can be placed, such that the x-Distance of v to any neighbor is at most k, for a fixed $k \in \mathbb{N}$. If we achieve this, then the resulting drawing Γ has maximum x-Distance $xs_\Gamma \leq k$. Specifically an interval of vertex v is defined as $I(v) = [p(r(v)) - k, p(\ell(v)) + k]$. Note that this interval is only well-defined for a large enough k. We observe that there exists a natural lower bound for the maximum x-Distance $k^\star$ which we denote as $k_0 := \lceil \frac{k_{\max}}{2} \rceil$, where $k_{\max} := \max_{v \in A}(p(r(v)) - p(\ell(v)))$, i.e. $k_{\max}$ coincides with the lower bound of the Window Width $ww_B(G)$. Clearly the x-Distance (in the critical part) of vertex $v \in A$ is optimal when placed exactly at the mean position of $\ell(v)$ and $r(v)$. See Figure 3.19 for the lower bound of the example graph. The algorithm will use this lower bound as an initial choice of the temporary maximum x-Distance $k := k_0$. Note that this k is large enough that all intervals are well defined. However in contrast to the proof of Theorem 3.1 the intervals are initially of smaller size. Incidentally there exists at least one interval consisting of at most two positions. While $2k_0$ is the largest possible initial size. Figure 3.20 shows the interval construction. As in the proof of Theorem 3.1, the algorithm might conclude that the current value of k is smaller than $k^\star$ while running, which implies that no drawing with maximum x-Distance $\leq k$ exists. Thus k is incremented before proceeding.

Observe that the algorithm provided for Theorem 3.1, after the initial interval construction, solely consists of assigning all vertices of A to positions within their respective intervals. Only increasing the interval sizes, when necessary. While the initial interval construction is different for this problem variation,

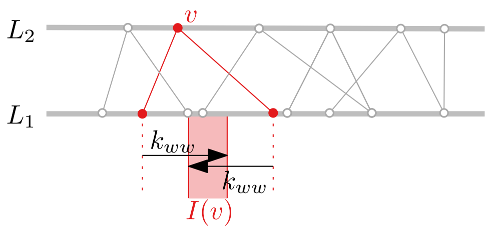


Figure 3.20: Showing the construction of an interval $I(v)$. While very similar to the interval construction in 3.1, the initial intervals differ significantly.

the subproblem is equivalent. Thus this part of the previous algorithm can be completely adopted. Also both the correctness and time complexity proofs of this algorithm follow analogously to the correctness and time complexity proofs of Theorem 3.1. □

3.5 x-Distance Sum Minimization with One Flexible Layer

Similar to the minimization of the maximum Window Width, minimizing the maximum x-Distance over all edges might not provide the desired result. While it does give a hard bound on the necessary screen size or resolution to display every edge completely on one screen, we might be inclined to accept a few long edges in practice, when we can achieve a much better result for the average edge. Particularly in the algorithm provided for Theorem 3.5 if there exists one edge prominently defining the maximum x-Distance, then all other edges tend to be equally long, as informally, the vertices of A are placed leftmost possible, making the edges to their rightmost neighbor often of length (close to) $xs_B(G)$. This is also clearly visible in the experimental results in Chapter 4. Therefore we investigate the natural variant of minimizing the average x-Distance of all edges. Note that this problem is equivalent to the problem of minimizing the sum of x-Distance. Also it is closely related, but not equivalent to the average edge length minimization, which is a known problem in Graph Drawing [10, 102]. Observe that, in contrast to Theorem 3.5, there is no simple correlation between the minimum sum of edge lengths and the minimum sum of x-Distances, unless the distance between top and bottom layers is 0.
As the problem is indifferent of interchanging A and B, we observe the problem where w.l.o.g. the vertices of B are assigned to fixed positions, while the vertices of A can be freely assigned. Note that if the vertices of A are fixed, then the vertex sets A and B can be interchanged.

Theorem 3.6. *Given a bipartite graph $G = (A \cup B, E)$ as well as a partial vertex placement $p\colon B \to \mathbb{Z}$. A 2-layer drawing Γ of G where the sum of x-Distances of all edges is minimized, while preserving the given vertex placement, can be constructed in $O(n_A^3 + m)$-time.*

Proof. The structure of this algorithm follows closely the structure of the proof of Theorem 3.2, with some small but significant changes. The problem instance can be expressed as a bipartite matching problem, which then can be solved by

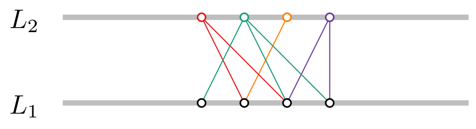


Figure 3.21: The running example with random vertex placements of the vertices of A in L_2 on consecutive coordinates. For this drawing $xs_\Gamma = 9$.

the Hungarian algorithm, finding a minimum weight maximum matching [109]. Note that in contrast to the algorithm of Theorem 3.2 the problem can not be reduced to a critical part, as all edges have to be considered. Thus, the first step of the algorithm is to construct a weighted bipartite graph $G^* = (A^*, B^*)$. See Figure 3.21 for the running example.

We make two important observations: the optimal placement of a vertex $v \in A$ is at the median of the positions of its neighbours $\{p(b)|(v,b) \in E\}$. This is true, as whenever placing v at x-coordinate x, when moving v by one to position $x+1$ $(x-1)$, all edges incident to v where the other endpoint is at position $\leq x$ increase (decrease) their x-Distance by exactly one. Analogously, all incident edges where the other endpoint is at position $> x$ decrease (increase) their x-Distance by exactly one. Therefore the summed x-Distance of incident edges of v is optimal, when v is positioned such that the number of neighbors positioned left of v is equal to the number of neighbors positioned right of v, thus exactly at the median of the positions of neighbors. Let m_v denote the median x-coordinate of $N(v)$. Further, observe that it suffices to consider a limited number of positions for placement of any vertex v. Specifically, the extended interval $[\lfloor m_v \rfloor - n_A, \lceil m_v \rceil + n_A]$ describes the set of positions to consider. We know that the sum of the x-Distances of all edges incident to vertex v increase monotonically with the distance of v from m_v. This follows from the observation of the optimal placement. Let $N_\ell(x)$ denote the set of neighbors of v which are positioned left of x and $N_r(x)$ the set of neighbors positioned right of x. At the median it holds that $N_\ell(m_v) = N_r(m_v)$, however for all $x < m_v$ $(x > m_v)$ it holds that $N_\ell(x) \leq N_r(x)$ $(N_\ell(x) \geq N_r(x))$. However at each increment, moving x further from m_v the sum of x-Distances of incident edges is increased by $|N_\ell(x) - N_r(x)| \geq 0$, therefor the sum increases monotonically.
Note that there are only $n_A - 1$ other vertices of vertex set A to be placed in total, however the best n_A positions of v regarding the summed x-Distances are included in the extended interval. Thus positions outside of the interval yield a solution which is no smaller than the one found by the algorithm. Further the extended interval $[\lfloor m_v \rfloor - n_A, \lceil m_v \rceil + n_A]$ consists of exactly $2n_A + 1$ or $2n_A + 2$ positions, thus they are bound in size.

Constructing G^*. The construction of the node and edge sets of G^* is analogously to the construction in the proof of Theorem 3.2, i.e. each position x, which is part of at least one interval is represented by node v_x in A^*. Also each vertex v of A is represented by a copy denoted v^* to B^*. Further, $\forall v_x \in A^*, v^* \in B^*$: $e = (v_x, v^*) \in E^*$ exactly if $x \in [\lfloor m_v \rfloor - n_A, \lceil m_v \rceil + n_A]$. An edge $e = (v_x, v^*)$ is weighted by the accumulated x-Distances of all edges incident to v, when placing v at position x.
This can be efficiently calculated by placing the neighborhood of v in an array of

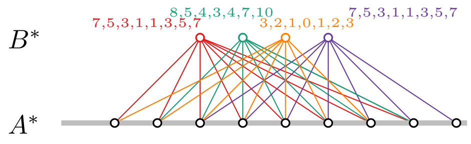


Figure 3.22: G^* for the summed x-Distance minimization of the running example. The edges weights are shown above the vertices of B^* in left to right order.

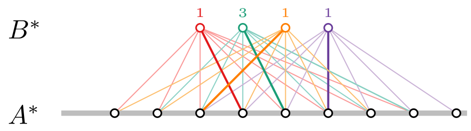


Figure 3.23: M^* for the summed x-Distance minimization of the running example. The corresponding edges weights are shown. This directly implies the optimal placement of the vertices of A.

size $2n_A + 1$, if there is an odd number of neighbors, or $2n_A + 2$, if the number of neighbors is even, according to the x-coordinates relative to $[\lfloor m_v \rfloor - n_A, \lceil m_v \rceil + n_A]$. If the x-coordinate of a neighbor are not within the interval, then the neighbor is considered to be permanently left or right of v. There are exactly $deg(v)$ neighbors to consider. Then we traverse the array keeping counters for the number of neighbors left ($N_\ell(x)$) and right ($N_r(x)$) to the current position x, the weights between two positions change by $N_\ell(x) - N_r(x)$. Deriving $N_\ell(x+1)$ and $N_r(x+1)$ from $N_\ell(x)$ and $N_r(x)$ can be done in constant time by checking the entries of the array at position x and $x + 1$. Thus in total $\mathcal{O}(n_A + \deg(v))$ time suffices to calculate the weights of all incident edges of one node in B^*. Figure 3.22 shows the constructed G^* for the running example.

Now that G^* is constructed, we compute a minimum weight maximum matching M^* of G^*. Since the degree of any node of B^* is exactly $2n_A + 1$ or $2n_A + 2$ and $|B^*| = n_A$, it holds that $|M^*| = n_A$, matching all nodes of B^*. If an edge $e = (v_x, v^*) \in M^*$, this corresponds to placing v at x-coordinate x. The weight of e correspond exactly to the sum of the x-Distances of incident edges. Thus, M^* corresponds to the optimal assignment of the vertices of A to unique x-coordinates minimizing the sum of x-Distances of all edges. Figure 3.23 shows M^* for the running example and Figure 3.24 the resulting optimal drawing.

Correctness. Clearly M^* corresponds to a valid position assignment of the vertices of A, as due to M^* being a maximum matching, any vertex of A is assigned exactly one position and every position is assigned to at most one vertex of A. Now, assume that there exists an assignment of the vertices of A

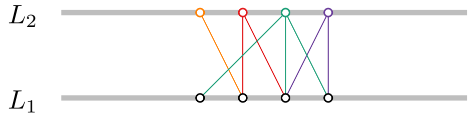


Figure 3.24: The optimal 2-layer drawing of G with fixed positions of B, minimizing the sum of x-Distances. We can see that $xs_B(G) = 6$.

to positions, such that the sum of the x-Distances is smaller than the sum of x-Distances in the algorithm solution. This optimal positioning corresponds to a matching M' on G^* which consists of n_A edges. By the assumtion and the definition of the weight function it must hold that $w(M') < w(M^*)$. However this is a contradiction, as M^* is the minimum weight maximum matching of G^*.

Time complexity. Constructing G^* can be done in $O(n_A^2)$ time, as $|E^*| \in \mathcal{O}(n_A^2)$ due to the interval definition. Computing the edge weights of all edges incident to a node $v^* \in A^*$ can be done in $\mathcal{O}(n_A + \deg(v))$ time. Thus, over all vertices of A this yields $\mathcal{O}(n_A^2 + m)$ time. Using the Hungarian algorithm to compute the minimum weight maximum matching takes $\mathcal{O}(n_A^3)$ time. Thus in total, the algorithm has a total time complexity of $\mathcal{O}(n_A^3 + m)$. $\square$

Chapter 4

Experiments

4.1 Experimental Setup

We proceed to implement these algorithms and apply them on randomly generated graphs. We study the difference on the respective metric between randomized and optimal placement. Our intend is to evaluate for which graphs the Window Width, Window Width sum, x-Distance and x-Distance sum are suitable optimization criteria and whether parameters exist for which there is no significant improvement between randomized and optimal placements. We use several well established graph generation algorithms to create the sample set, varying the size and density of the graphs. To this end the number of vertices of A is denoted as n_A, the number of vertices of B as n_B and the expected number of edges as m^*. First we shortly introduce the graph generation models used.

4.1.1 The Gilbert Graph Model

We denote the random graph model first described by Edgar Gilbert [63] as the *Gilbert* model. In the Gilbert model two parameters n and p are taken as input, $n \in \mathbb{N}$ corresponds to the total number of vertices, while $0 < p < 1$ gives the independent probability for any edge to exist in the randomly sampled graph. Thus, the number of edges m varies, and any one specific graph with m edges has total probability of $p^m(1-p)^{\binom{2}{n}-m}$[29].
As we are interested in generating bipartite graphs, this model has to be adjusted accordingly. To do so, we take the two parameters n_A and n_B instead of n as input. The probability of an edge to exists is still independently defined by p, however restricted to bipartite edges. Thus, edges with both endpoints in either A or B have probability 0. The expected number of edges of a randomly generated graph with fixed parameters n_A, n_B and p is $p \cdot n_A \cdot n_B$. As we have a given expected number of edges m^* as input, we define $p := \frac{m^*}{n_A \cdot n_B}$ accordingly.

4.1.2 The Erdős–Rényi Graph Model

The *Erdős–Rényi* model [53] takes two parameters n and m as inputs where n is the number of vertices and m the number of edges. The graph is chosen uniformly out of the set of all graphs satisfying both n and m. Such a graph can be generated by initially starting with n vertices and no edges and iteratively adding an edge, which is uniformly chosen out of the set of all edges not in the edge set of the graph. In total m edges are selected. Again, as we are studying algorithm on bipartite graphs, we have to adjust the model. Thus we take two parameters n_A and n_B instead of n as input. The iterative steps can be performed analogously to the general case, however the edges are selected from the edge set only containing bipartite edges, i.e. edge with one endpoint in A and one in B. By setting $m := m^*$, all generated graphs have exactly the expected number of edges.

4.1.3 The Barabási-Albert Graph Model

The *Barabási-Albert* model aims to model natural and human-made networks. Those kind of networks are not satisfyingly modelled by the previous graph generation models [1]. To achieve this, the model uses the preferential attachment method, where edges are more likely to be connected to vertices which already are of high degree. In contrast to the other discussed models a power-law distribution of the vertex degrees is achieved, replicating that of most real-world networks. The algorithm takes two parameters $n \in \mathbb{N}$ and $k \in \mathbb{N}, k \leq n$, as inputs. It starts with a minimal connected vertex set consisting of at least k vertices and then iteratively adds vertices until the total number of vertices reaches n. When adding a new vertex v, a fixed number of k incident edges are introduced simultaneously. The edges are not distributed uniformly, instead the probability that an edge is incident to vertex w is defined as

$$p_w = \frac{deg_w}{\Sigma_{u \in V'} deg_u}$$

where V' denotes the set of vertices, which were already introduced. As before, the algorithm has to be adjusted to generate bipartite graphs. To this end we generate the initial graph by creating a path of $2k$ alternating between vertices of A and B.
For the iterative construction as before one vertex is added at a time until there are n vertices in total. However when adding a vertex v we randomly decide, whether v is added to A or B. A Bernoulli trial is used for that decision, where the chances are $\frac{n_A-k}{n_B-k}$, so that the expected number of vertices assigned to A and B corresponds to n_A and n_B. In total $n-2k$ vertices are added to the initial graph. W.l.o.g. assume that v is assigned to B, then the preferential attachment method is used to distribution the incident edges of v, however using the probability distribution

$$p_w = \frac{deg_w}{\Sigma_{u \in A'} deg_u}, \forall w \in A'$$

where A' is the set of all vertices of A, which were already introduced.
Note that for a fixed k, every iteratively added vertex introduced k new edges, thus the total number of edges is $k \cdot (n_A + n_B - 2k) + 2k - 1$. Since k has to be a natural number, the number of total edges can diverge from m^* due to rounding.

4.1.4 Sampling

Observe, that graphs generated by the Gilbert and the Erdős–Rényi model can contain isolated vertices. The chance for isolated vertices increases for lower m^* and higher total number of vertices. The Barabási-Albert model however does always generate a connected graph.
If there exist isolated vertices after a graph generation, these are removed in a preprocessing step as their placement is independent of the studied metrics (they have no neighbors and no incident edges).

For the experimental evaluation, graphs of different sizes are generated. Specifically $n_A \in [10, 50, 100]$ and vertex set B is chosen relative to n_A, specifically $n_B \in [0.5n_A, n_A, 2n_A]$. The edge densities are chosen according to linear, logarithmic and quadratic magnitudes. In detail, for linear many edges $m^* = 2(n_A + n_B)$, for a logarithmic number of edges $m^* = \log(n_A + n_B)(n_A + n_B)$ and for a quadratic density $m^* = \frac{n_A \cdot n_B}{4}$ edges are expected. For each tuple of parameters, we randomly sample 1000 graphs according to each random graph model, which are then evaluated according to the metrics. First we evaluate the initial drawings, where the vertex positions are assigned consecutively in a random permutation. We remark that if vertex sets A and B are of different cardinality the vertex placements are centered, i.e. both A and B have the same average x-coordinate. We evaluate the metrics for the these initial drawings.
We then apply the respective optimization and approximation algorithms and evaluated the metric for the resulting drawing.

We evaluated the Window Width, the x-Distance, the Window Width sum and the x-Distance sum before and after optimization.

4.2 Experimental Results

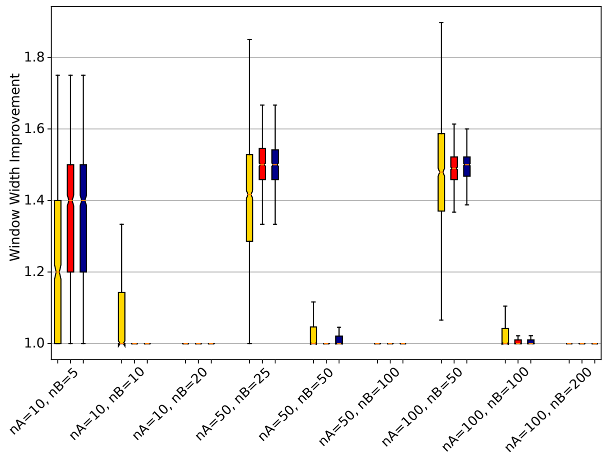
We discuss findings for the four different metrics in the following, drawing a conclusion for each.

Window Width The experimental results for the Window Width before and after optimization are shown in Figure 4.1. As expected, the variance decreases with increasing graph size. The expected improvements in Window Width also clearly correlates with the ratio n_A to n_B. If $n_B = 2n_A$ almost no improvement is visible, when optimizing the Window Width over random vertex placement. The Window Width can only improved if for the largest Window Width (at vertex v) at random vertex placement, both $\ell(v)$ and $r(v)$ are to the same side of v, however for larger graphs, this gets increasingly unlikely.
The biggest improvements were possible for graphs where $n_A = 2n_B$, with an average improvement of approximately 1.4. The differences between graph generation models is minor except for the variance, which is significantly higher for Gilbert graphs. The edge density seems to correlate negatively with the measured improvement, noticeably for quadratic densities, where the improvement decreased for larger graphs.
Overall, we find that significant improvements are possible if there are more vertices of n_A than of n_B, otherwise there are likely other optimization criteria with greater impact on the readability.

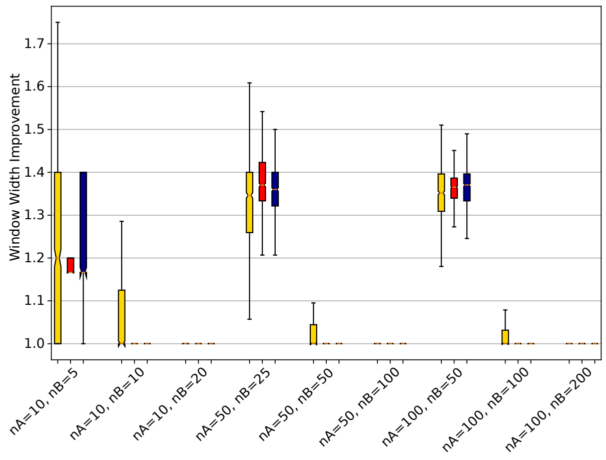
Window Width Sum The experimental results for the sum of all Window Widths before and after optimization are shown in Figure 4.2. We observe similar behavior as for the maximum Window Width. However, while the improvement is still reduced, when the ratio n_A to n_B decreases, the effects are not nearly as extreme. Overall the average improvement is approximately 1.2. While this is lower than the improvement of maximum Window Width for graphs with $2n_A = n_B$, it is significantly higher for other ratios. Again, a negative correlation with the graph density is visible. Surprisingly, for larger graphs with quadratic density very little improvement is possible. While for a single edge the same argument as for the maximum Window Width holds, considering the sum of Window Widths, the constellation, where the random placement of v is already in between $\ell(v)$ and $r(v)$ has to happen for all vertices $v \in n_A$ independently. However with a large enough average degree this becomes increasingly likely.
Overall, optimization decreases the Window Width sum significantly for most instances. Only for high edge density and relatively large graphs, the improvement becomes insignificant compared to random vertex assignment.

x-Distance The experimental results for the maximum x-Distance before and after optimization are shown in Figure 4.3. Again, as expected, the variance decreases with larger sizes of graphs. Overall the x-Distance improvement for linear edge density is higher than for denser graphs, reaching up to 2.1 on average for graphs with $n_A = 100$ and $n_B = 50$. There is only a weak negative correlation with the ratio n_A to n_B. For quadratic edge density the x-Distance improvement is reduced with increasing graph size. Generally, it appears that for large enough graphs the improvements for graphs generated by the Gilbert model are slightly but consistently greater compared to the other models.
Overall it appears that optimization improves the x-Distance significantly for all graphs with sparse to logarithmic density. For quadratic edge density, an optimization still strongly affects the x-Distance if the graph is relatively small.

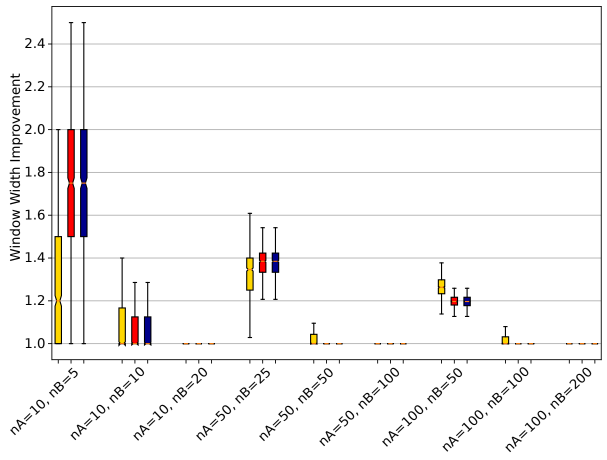
x-Distance sum Lastly, the experimental results for the x-Distance sum before and after optimization are shown in Figure 4.4. We observe very similar behavior as for the maximum x-Distance. There is a correlation between the improvement and the ratio n_A to n_B. With logarithmic and quadratic density the improvement is reduced for larger graphs. The comparison between graph generation models is less clear here, graphs generated by Erdős–Rényi and Barabási-Albert behave almost identical, graphs generated by the Gilbert model differ, however depending on the graph size and the density they yield a greater or smaller improvement. Overall the best improvement is possible for graphs with linear density, $n_A = 100$ and $n_B = 50$. Again, for graphs of quadratic density and increasing graph size the x-Distance sum for an optimal and a random vertex placement seem to converge.
Overall optimizing the x-Distance sum is significant for most instances. Only large graphs with high density can only be minorly improved by optimization.



(a) Linear density

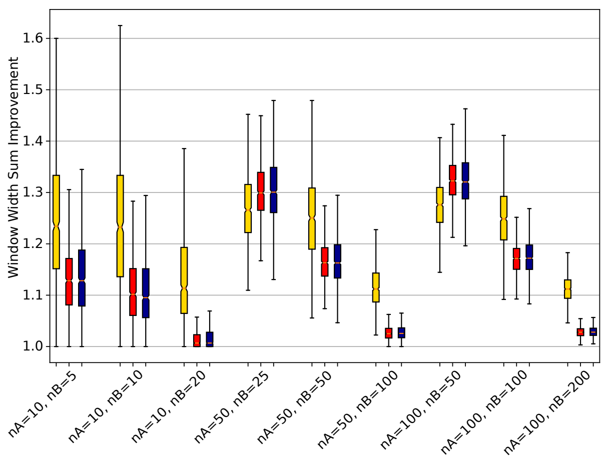


(b) Logarithmic density

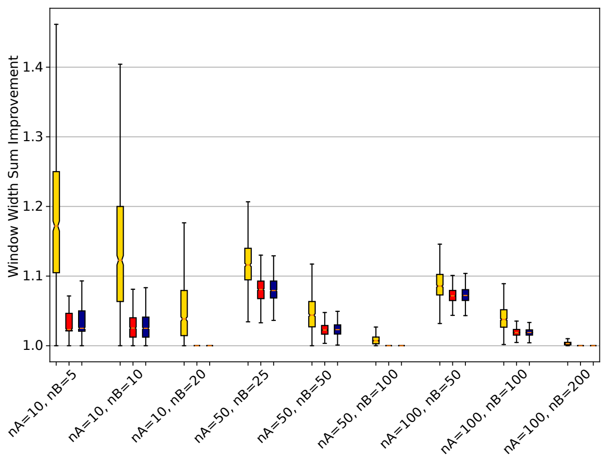


(c) Quadratic density

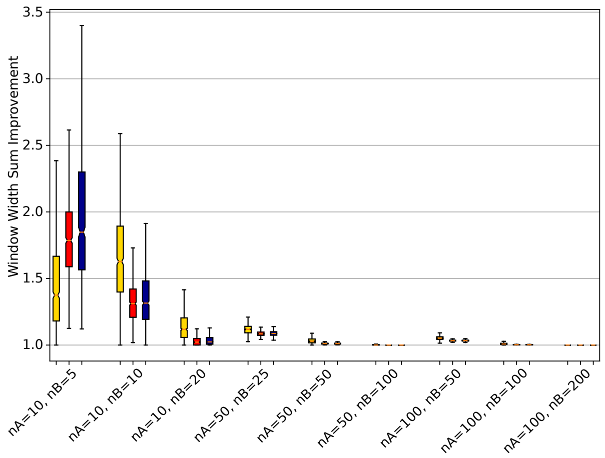
Figure 4.1: The plots in (a), (b) and (c) show the improvement of the Window Width on graphs generated with the respective models. Gilbert graphs are shown in yellow, Erdős–Rényi in red and Barabási-Albert in blue.



(a) Linear density

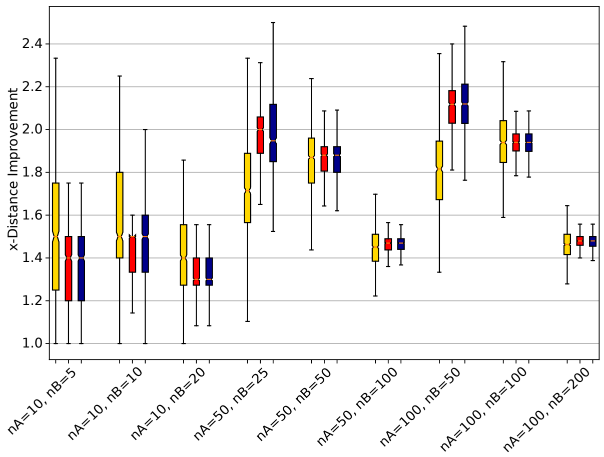


(b) Logarithmic density

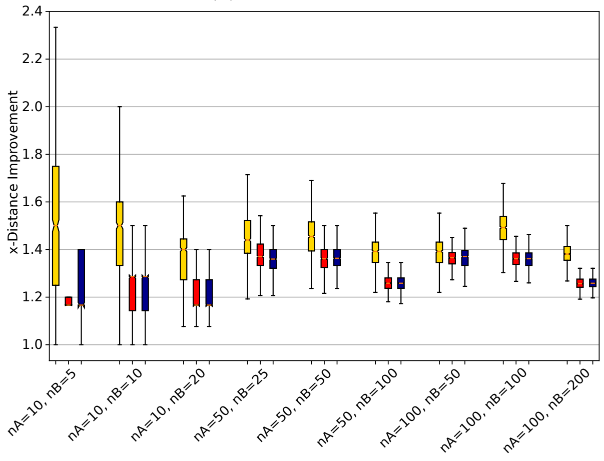


(c) Quadratic density

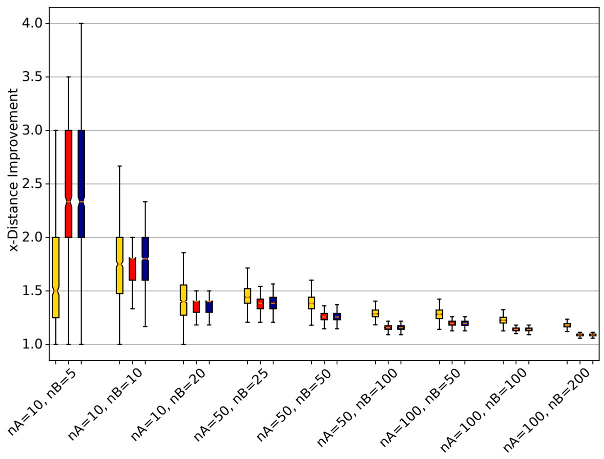
Figure 4.2: The plots in (a), (b) and (c) show the improvement of the Window Width Sum on graphs generated with the respective models. Gilbert graphs are shown in yellow, Erdős–Rényi in red and Barabási-Albert in blue.



(a) Linear density

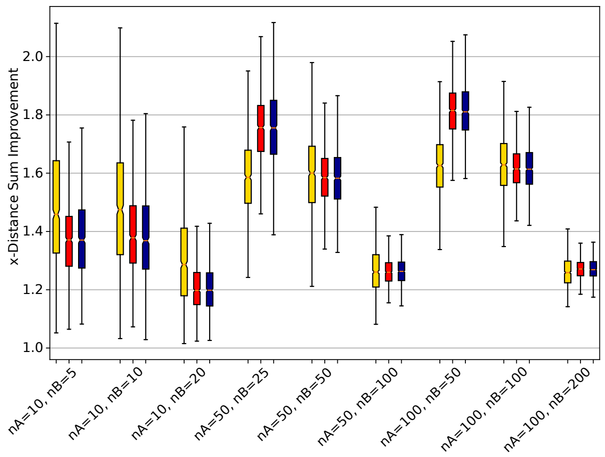


(b) Logarithmic density

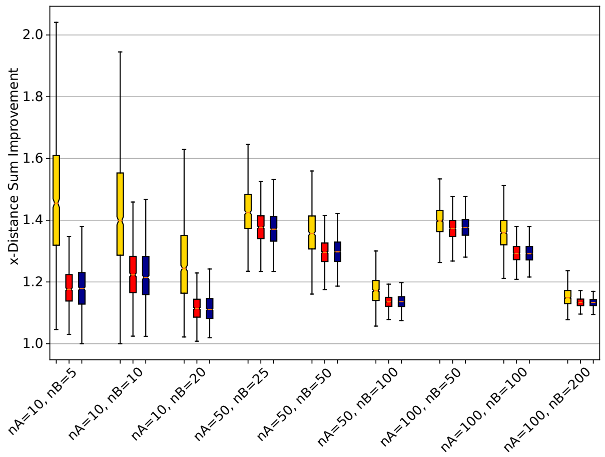


(c) Quadratic density

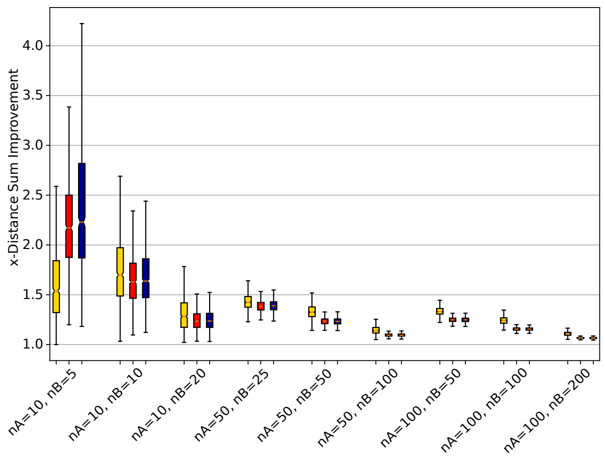
Figure 4.3: The plots in (a), (b) and (c) show the improvement of the x-Distance on graphs generated with the respective models. Gilbert graphs are shown in yellow, Erdős–Rényi in red and Barabási-Albert in blue.



(a) Linear density



(b) Logarithmic density



(c) Quadratic density

Figure 4.4: The plots in (a), (b) and (c) show the improvement of the x-Distance Sum on graphs generated with the respective models. Gilbert graphs are shown in yellow, Erdős–Rényi in red and Barabási-Albert in blue.

Part II

Simultaneous Embedding of Multiple Upward Trees

Embedding of Multiple Upward Trees

In this part of the thesis we will study rooted trees with an upward layered drawing, where a total order of all leaves is given. The problem consists of finding an ordering of the vertices of all other layers such that the resulting drawing is crossing minimal. The problem is motivated by having multiple disjoint hierarchical structures, such as phylogenetic trees or organizational diagrams, which, however, are shown intertwined.
Most hierarchical structures take the form of trees, this is the case, whenever one object has a unique affiliation. For instance this is true for most business hierarchies as any employee has one direct supervisor. It is also true for evolutionary taxonomy, where no two species can be ancestor of another species.
Thus, it is not surprising that a diverse set of tools for the visualization of trees exists. However most of these techniques assign vertex positions freely, such that a adequate representation is achieved. For multiple hierarchical structures at once, this usually means placing them side by side, to avoid any edge crossings. However practical applications arise, where the visualization imposes requirements on the vertex placement. For instance if species should be ordered alphabetically, to allow for efficient look-up, or employees which should be are arranged by income. This requires a fixed order of all leave vertices. Still the hierarchical structure above should be maintained and be easily readable. Observe that due to the additional requirement, a crossing free representation is generally no longer possible. Thus, we aim to find a representation which minimizes the number of edge crossings, which is known to be a good indicator for the readability of a drawing [106].
Note that the problem closely relates to the Sugiyama framework [126], which is a state-of-the-art method of visualizing general hierarchical system structures, usually while minimizing the number of crossings. However without any restriction on the graph structure, most application of the Sugiyama framework rely heavily on heuristics. This is because finding a crossing minimal solution is NP-hard, even on a layer by layer approach, where only two consecutive layers are considered and the vertex placement of one is assumed to be fixed. However utilizing the fact that our graph is a forest, we can achieve more promising results. For this purpose we introduce the *Crossing Minimization of Upward Trees* or CMUT problem as follows: Given a forest F of k rooted trees $T_1, \ldots, T_k$ with all leaves in layer L_1, each with a layered upward embedding $\varepsilon_1, \ldots \varepsilon_k$, as well as a fixed order of all leaves in L_1 denoted as $\prec_1$. What is the minimal number of crossings to draw F respecting $\varepsilon_1, \ldots \varepsilon_k$ and $\prec_1$. Figure 5.1 shows an exemplary problem instance.

It is a known result, that this problem is NP-complete for an arbitrary number of trees, even on just two layers, when a total order of the leaves is given [95]. Thus, we split this part into two chapters, where Chapter 6 restricts the problem to two trees, presenting a polynomial time optimization algorithm as well as results for some variations of the problem. Chapter 6 studies the problem for any number of trees, however on 2 and 3 layers only. As the problem is NP-complete for an arbitrary number of trees, an FPL-algorithm will be presented, for the case of two layers and an XP-algorithm if the number of layers is restricted to three. Some further results considering variations of the problem are also

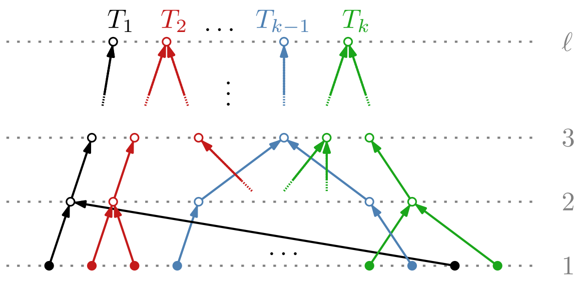


Figure 5.1: Upward drawing of k directed rooted trees $T_1, \ldots T_k$ on ℓ layers. The total order of layer one is given, indicated by the filled disks, while there are no (partial) orders for layers 2 and 3. In the following figures, arrow heads are omitted as an upward direction can be assumed.

discussed. Note that there is a gap in between the two (main) results, which remains an open problem.

Some results of this chapter also appeared in [83][1].

[1]All collaborators contributed to equal parts to the results of this paper.

Chapter 6

Limited Number of Trees

We start by studying the case where the given forest F consists of only two rooted trees, since we denote the number of trees in F as k, we assume that $k = 2$ for this chapter. We denote the two trees as T_1 and T_2. Their roots are called r_1 and r_2 respectively. All leaves are on the same, bottommost layer L_1, which we will also refer to as the *leaf layer*. Layered upward planar drawings $\varepsilon_1, \varepsilon_2$ are given for both T_1 and T_2 respectively, where vertices are assigned to a layer. The (arbitrary) maximum layer out of both drawings will be denoted as layer ℓ or L_ℓ. We will assume that there exist no edges spanning more than one layer, i.e. for any non-root vertex v in T_1 or T_2, it holds that, if v is in layer L_i, then its parent must be in layer $L_i + 1$. If a graph has edges spanning more than one layer, dummy vertices can be introduced in a preprocessing step to subdivide the edges such that the resulting edges span exactly one layer each. Note that planarity can easily be maintained during this preprocessing step and the subdivisions are unique. Let n_1 and n_2 denote the number of vertices of T_1 and T_2 respectively after the preprocessing.

5.1 CMUT for two trees

In this section we will present a polynomial time algorithm optimally solving the CMUT-problem for two trees. However to show its correctness, we first need to introduce some notation and discuss preliminary results.
First we observe that it is sufficient to add the non-leaf vertices of T_2 to the drawing of T_1 given by ε_1 such that the number of crossings is minimized, thus, in the following we will assume the drawing of T_1 to be fixed and only construct an according drawing of T_2.

Consider layer $j \in \{2, \ldots, \ell\}$ of the graph. $V_j(T_1)$ denotes the set of vertices of T_1 in layer L_j. In the following we will assign all vertices of $V_j(T_1)$ an integer value such that they are indexed from left to right, i.e. numbers $1, \ldots, n_{1|j}$ are assigned, where $1|j$ is denotes the number of vertices of T_1 in layer j. Now the placement of a vertex of T_2 can be described by its position relative to the vertices of T_1. For a vertex $v \in V_j(T_2)$, its *position* p is defined by the index of the closest vertex $w \in T_1$ to the left of v. Note that it is possible, that no such

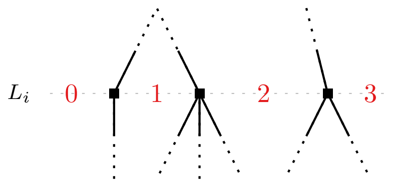


Figure 5.2: An example showing the positions 0 to $1|j$ in layer j in red. The positions are defined according to the next vertex of T_1 to the left.

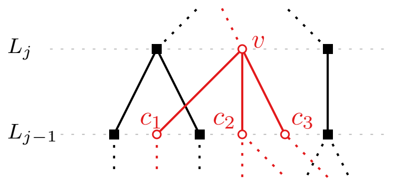


Figure 5.3: An example showing the nomenclature of the children C_v of $v \in T_2$, ordered left to right. They are indexed in anti-clockwise order. all vertices of C_v are in layer $j-1$.

w exists, in which case $p := 0$. Observe that the position corresponds to the number of vertices of T_1, which are left of v in layer j. Figure 5.2 illustrates this definition with an example.
Further let $C_v = (c_1, c_2, \dots, c_{|C_v|})$ denote the set of children of vertex v, ordered by their left to right positions. Necessarily these vertices are in layer $j-1$. Figure 5.3 provides an example.

We solve the CMUT problem for two trees using a dynamic program, which constructs optimal partial solutions for any subtree of T_2 rooted at vertex v for any position p of v. To achieve this we define a function o which takes vertex v as first and position p as second parameter and returns the number of crossings of an optimal partial solution, assuming that v is placed at p. According to the dynamic algorithm principle these optimal partial solutions are calculated only once by considering the layers in a bottom-up fashion and storing the partial results in a look-up table. As usual for dynamic algorithms a witness can be constructed by back-tracking. To simplify this final step we save the recursive dependency in the bottom-up pass. Thus, a witness drawing can be constructed with a single top-down pass at the end. For $j \geq 3$, o is defined as follows.

$$o[v,p] = \sum_{c_i \in C_v} min_{q \in \{0,\dots,n_{1|j-1}\}}(o[c_i,q] + cr_{j-1}(q,p))$$

where $cr_h(y,z)$ denotes the *crossing function* which is defined as the number of crossings of an edge e spanning between layers h and $h+1$ if the endpoint of e in layer h is placed at position y and the other endpoint in layer $h+1$ at position z. See Figure 5.4 for an example.
The function thus consists of a sum, where each summand corresponds to one child vertex c_i. Then, the optimal position q for p_i is selected and the resulting number of crossings are added up. These crossings comprise of the crossings of the subgraph rooted at c_i, explicitely $o[c_i,q]$ and the crossings of the edge (v,c_i), which is accounted for in $+cr_{j-1}(q,p)$.

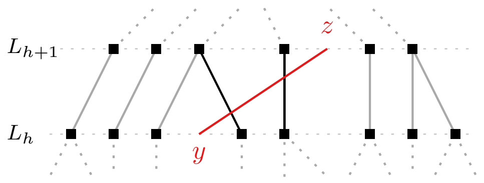


Figure 5.4: Example of the crossing function $c_h(y, z)$ returning the number of crossings an edge from position y in layer h to position z in layer $h + 1$ would yield.

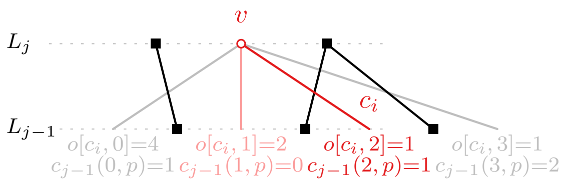


Figure 5.5: Example illustrating the tie-breaker strategy. When computing $o[v, p]$, all children of v are considered separately. Regarding child c_i of v all positions where $o[c_i, q] + cr_{j-1}(q, p)$ is not minimal are greyed out, they cannot result in an optimal (partial) solution. Using the tiebreaker rule, among the remaining solutions the one is chosen, where $cr_{j-1}(q, p)$ is maximal, marked in dark red.

To finalize the recursive function we define the case for layer $j = 2$. As the order $\prec_1$ of all leaf vertices is fixed, the position of all children of any vertex $v \in V_2(T_2)$ is fixed. Thus, we obtain the following simplified case.

$$o[v, p] = \sum_{c_i \in C_v} c_1(p_{c_i}, p))$$

p_{c_i} denotes the position of child vertex c_i as defined by $\prec_1$.

To compute the total number of crossings o^* of the optimal solution it suffices to find the optimal placement for the root of T_2, i.e. r_2, using the pre-calculated partial solutions.

$$o^* = min_{p \in \{0,\ldots n_{1|L(r_2)}\}} o[r_2, p]$$

It only remains to find a witness, by constructing a drawing corresponding to o^*. As stated before, we store the recursive dependencies during the bottom-up computation. However we implement an important tie-breaker rule. Specifically, when computing $o[v, p]$, if for some child vertex c_i of v there exist multiple positions $\{q_1, \ldots, q_r\}$ which minimize $o[v, p]$, i.e. $o[c_i, q_i] + cr_{j-1}(q_i, p)$ is minimal for all $i \in \{1, \ldots, r\}$, position $q \in \{q_1, \ldots, q_r\}$ is chosen such that $cr_{j-1}(q, p)$ is maximized. Informally this corresponds to selecting the placement where crossings occur between the highest possible layers, while the global minimum is maintained. See Figure 5.5 for an example of this tiebreaker rule. With this tie-breaker rule, the positions of all vertices of T_2 yielding o^* crossings in total are defined. If multiple vertices of T_2 within the same layer j share the same position p, they are arranged (within the boundaries of the position) according to the order provided by ε_2. Figure 5.6 provides a full example of the algorithm application on a problem instance.

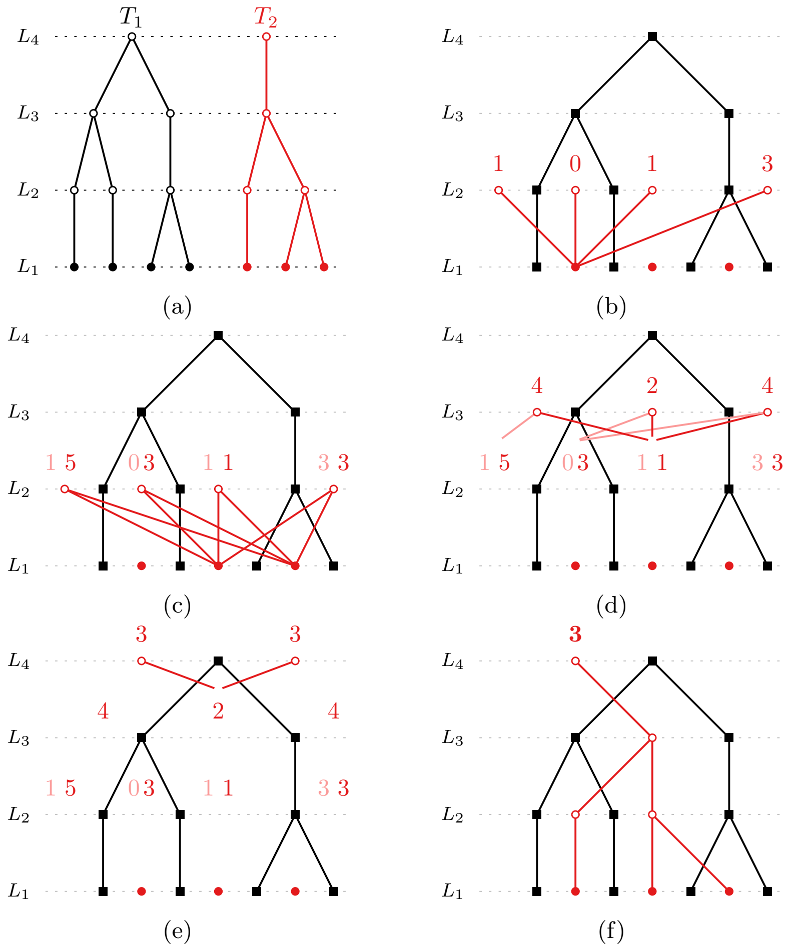


Figure 5.6: A full example applying the dynamic program. Subfigure (a) shows ε_1 and ε_2 side by side. Subfigure (b) shows the optimal partial solutions of the left vertex of T_2 in layer 2. Note that its only child has a fixed position as it is a leaf. Subfigure(c) shows the optimal partial solutions of the right vertex of T_2 in layer 2, the optimal partial solutions of the left vertex are still depicted in light red. Subfigure (d) shows the optimal partial solutions of the only vertex of T_2 in layer 3, utilizing the optimal partial solutions of its children. Note that the crossings of the partial solutions and the crossings between layers 2 and 3 have to be added. Subfigure (e) shows the same for the root r_2. In (d) the witness drawing with minimal number of crossings is constructed. There are 3 crossings in total.

Before proving the correctness, we shortly analyze the running time of the algorithm.

Lemma 5.1. *The algorithm can be applied in $\mathcal{O}(n_1^2 \cdot n_2) \in \mathcal{O}(n^3)$ time.*

Proof. First we can pre-compute all values of $cr_j(q,p)$ in quadratic time $\mathcal{O}(n_1^2)$ as any layer consists of at most n_1 positions. To do this efficiently assume p is fixed. There exists one p^* for which $cr_j(q,p^*) = 0$, the crossings for all other positions p can now be easily computed as $cr_j(q,p) = |p - p^*|$, in constant time each, see Claim 5.1. Thus, overall $\mathcal{O}(n_1^2)$ is obtained.
Clearly, there exist overall $\mathcal{O}(n_1 \cdot n_2)$ entries of $o[v,p]$. Observe that to compute $o[v,p]$ the position q of each children $c \in C_v$ is determined individually, such that $o[c,q] + cr_{j-1}(q,p)$ is minimized. Thus, every entry is accessed $\mathcal{O}(n_1)$ times, once for all positions of its parent. As the crossings are already pre-computed, computing all entries $o[v,p]$ takes total time $\mathcal{O}(n_1^2 \cdot n_2)$. Selecting the optimal root placement takes linear time $\mathcal{O}(n_1)$ as there $\mathcal{O}(n_1)$ positions of the root r_2. Since each entry $o[v,p]$ already sets a pointer to the optimal partial solutions of its children at computation, construction of the witness drawing can be efficiently done in linear time.
Thus, the overall complexity is $\mathcal{O}(n_1^2 \cdot n_2) \in \mathcal{O}(n^3)$. Note that if the sizes of both trees differ significantly, interchanging T_1 and T_2 can improve the running time. □

Now we prove the correctness of the given algorithm. First we make an essential observation. Due to the tree structure of T_1, for position p on layer j there exists exactly one position p^* for which $cr_{j-1}(p^*,p) = 0$ holds, i.e. if an edge of T_2 is drawn with endpoints placed at p and p^* respectively, the edge is crossing free. We denote such a position p^* as *ideal*. Further, if we observe two positions p and q, both on layer j and $p < q$ holds, then it also holds that both ideal positions p^* and q^* of p and q respectively are ordered analogously: $p^* < q^*$ on layer $j-1$. Informally p and q correspond to faces (however these faces are open at the leaf layer), p^* and q^* are placed in the same respective faces, although in layer $j-1$. Clearly this means there exist positions which are never ideal. Figure 5.7 illustrates the concept of ideal positions.
In the following claim, this observation is extended. For simplicity both $o[v,p]$ and cr_j are assumed to simply return ∞ if called with parameters outside of their domain.

Claim 5.1. *Let $p \in \{0,\ldots,n_{1|j}\}$ be a position on layer $j \in \{2,\ldots,L(r_1)\}$ and $p^* \in \{0,\ldots,n_{1|j-1}\}$ its corresponding ideal position on layer $j-1$. It holds that $cr_{j-1}(p^* \pm x, p) = x$ and $cr_{j-1}(p^*, p \pm (x+1)) > cr_{j-1}(p^*, p \pm (x)) \geq x$ for any $x \in \mathbb{N}_0$.*

Proof. Consider an ideally placed edge $e = (u,v)$ of T_2, where both incident vertices are assigned positions p and p^* respectively. As observed previously this edge is crossing free, in respect to T_1. Thus, $cr_{j-1}(p^*,p) = 0$. Now if the position of u is increased or decreased, for any incrementation (decrementation) the vertex u interchanges the order with one vertex w of T_1. As w cannot be the root, it has exactly one parent, i.e. there is exactly one incident upward edge of w which was uncrossed before but due to the interchange is now crossed. Thus, the number of crossings is increased by exactly one. This occurs for x vertices of T_1

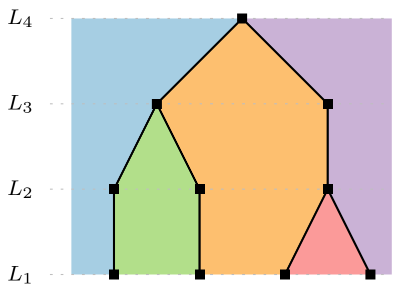


Figure 5.7: The areas corresponding to mutual ideal positions are marked by color for this instance. Specifically for any vertex v at a position of a given area, the ideal position of a child of v is within the same area on the layer below. We can see, that these areas closely resemble faces, however they are not closed at the leaf layer.

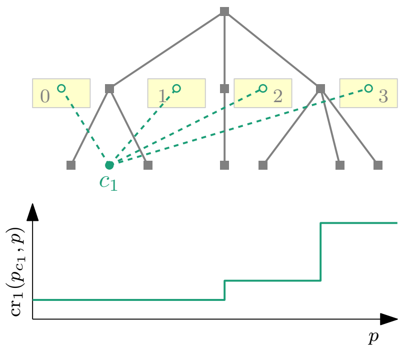
independently. Thus, there are exactly x crossings in total. Analogously, consider incrementing (decrementing) the position of v in layer j. Every incrementation (decrementation) interchanges v with one vertex of w. Since w cannot be a leaf, it has at least one child, which means at least one incident incoming edge. Thus, the number of crossings is increased by at least one, more precisely by $indeg(w) > 0$. Therefore in total the number of crossings is increased by at least x. By the same argument $cr_{j-1}(p^*, p \pm (x+1)) > cr_{j-1}(p^*, p \pm (x))$ holds, as there is one additional interchange, adding at least one crossing. $\square$

Considering vertex $v \in V_j(T_2)$ on layer j, we call the set of positions p where $o[v, p]$ is minimum as *optimal positions*, denoted $P_{opt}(v)$. Accordingly $\min P_{opt}$ denotes the leftmost vertex of the vertex set $P_{opt}(v)$. Analogously $\max P_{opt}$ denotes the rightmost position.

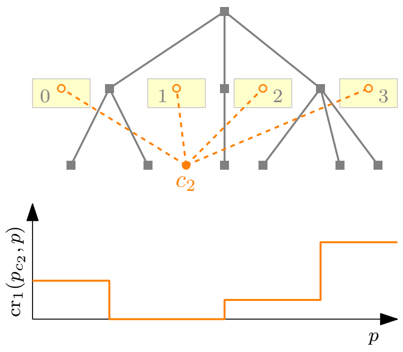
In the following we claim, that the optimal positions consist of a connected interval of natural numbers. Further, for positions outside of P_{opt} the number of crossings strictly increases with distance to the interval. Also we claim that the intervals are ordered left to right according to ε_2, however the intervals can overlap.

Claim 5.2. *Let $v_1, v_2, \ldots, v_{n_{2|j}}$ be the vertices $\in V_j(T_2)$ of layer $j \in \{2, \ldots, L(r_2)\}$ in order of ε_2. For every $v \in V_j(T_2)$, $P_{opt}(v)$ is an interval of natural numbers. For $x \in \mathbb{N}_0$ it holds that $o[v, \min P_{opt}(v) - (x+1)] < o[v, \min P_{opt}(v) - x] \geq x$ and $o[v, \max P_{opt}(v) + (x+1)] > o[v, \max P_{opt}(v) + x] \geq x$.*
Further $\min P_{opt}(v_1) \leq \min P_{opt}(v_2) \leq \cdots \leq \min P_{opt}(v_{n_{2|j}})$ and $\max P_{opt}(v_1) \leq \max P_{opt}(v_2) \leq \cdots \leq \max P_{opt}(v_{n_{2|j}})$ holds.

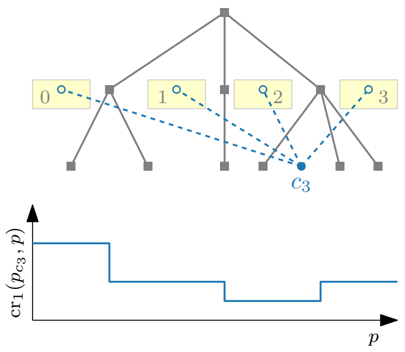
Proof. We prove the claim by induction over layers $j = 2, 3, \ldots$. For the base case $(j = 2)$, let $v \in V_j(T_2)$. We observe that all children of v have a fixed position as they are in the leaf layer. Therefore the partial optimal solution $o[v, p]$ is solely dependent on the number of crossings implied by position $p \in \{0, \ldots, n_{1|j}\}$ of v. Figure 5.8 provides an example. We continue with showing that $P_{opt}(v)$ is an interval. To do so, we observe that the function $o[v, p] = \sum_{c_i \in C_v} c_1(p_{c_i}, p)$ where p_{c_i} denotes the position of c_i in the leaf layer. Therefore it is the sum of $|C_v|$



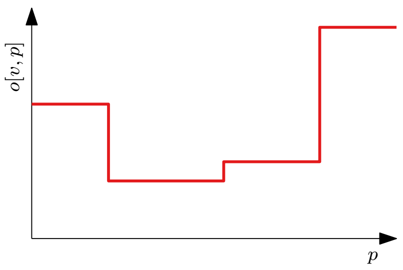
(a) The crossing numbers for a fixed leaf child c_1 with parent position p as variable.



(b) The crossing numbers for a fixed leaf child c_2 with parent position p as variable.



(c) The crossing numbers for a fixed leaf child c_3 with parent position p as variable.



(d) The resulting function $o[v, p]$, combining the functions (a)-(c) as summands, with position p as variable.

Figure 5.8: An example, depicting how $o[v, p]$ corresponds to a unimodal function with p as the variable, which is the sum of multiple unimodal functions.

many discrete functions, with p as a variable. Each of these functions admits its minimum for at most two neighboring positions. Further these functions are unimodal, i.e. they have exactly one global minimum and increase monotonously with increased distance to that minimum. Specifically, when incrementing or decrementing p by one, all functions either decrease or increase their value by the same amount, which is the number of children of the p-th vertex. Unless both p and $p+1$ $(p-1)$ form the minimum of the function.
To find the minimum of $o[v,p]$ we start at position $p=0$. Now if p is incremented, then all summand functions, which have not yet reached their minimum decrease by some fixed amount z. Analogously, all functions which already passed their minimum increase the sum by z. Thus, this sum is minimal exactly where the number of minimas of the summands are balanced, i.e. there are equally many minimas, weighted by the vertex degree, left and right of p. Naturally this forms an interval on the domain. Outside of this interval P_{opt}, for each position which is further distanced from the interval, the sum increases by at least one, as the number of minimas is inbalanced, which means the number of children of the p-th vertex of T_1 in layer 2 is added at least once. Since this vertex cannot be a leaf, it has at least one child. therefore $o[v, \min P_{opt}(v)-(x+1)] < o[v, \min P_{opt}(v)-x] \geq x$ and $o[v, \max P_{opt}(v)+(x+1)] > o[v, \max P_{opt}(v)+x] \geq x$ holds.

For the induction step $j > 2$. As previously, $o[v,p]$ corresponds to a sum, however the summands now consist of two parts, namely $cr_{j-1}(q,p)$ and $o[c,q]$ for child c of v at position q. Considering one such function we observe that this function is minimal for all positions p for which $p^* \in P_{opt}(c)$ and strictly increases monotonously outside of the interval, both due to the induction hypotheses. Thus, $o[c,q]$ increases monotonically if q is outside of $P_{opt}(c)$ and due to Claim 5.1 $cr_{j-1}(q,p)$ increases monotonically if q is not p^*. Thus, considering the sum, the previous argument holds, proving the properties of the claim. $\square$

We define a natural position denoted as $p_{nat}(u,p)$ for every vertex $u \in T_2 \setminus \{r_2\}$ and a position p of its parent vertex. The natural position is $\min P_{opt}(u)$ if $p^* < \min P_{opt}(u)$, it is p^* if $p^* \in P_{opt}(u)$ and it is $\max P_{opt}(u)$ if $p^* > \max P_{opt}(u)$. Informally the natural position is the position closest to p^* within P_{opt}.
We claim that for $v \in T_2$ at position p, placing all child vertices at their natural position yields the optimal solution for $o[v,p]$. Further we claim, that the natural positions of all children of v are ordered as in ε_2.

Claim 5.3. *Considering a vertex $v \in V_j(T_2)$ on position $p \in \{0,\dots,n_{1|j}\}$ in layer j. It holds that $o[v,p] = \sum_{c_i \in C_v}(o[c_i, p_{nat}(c_i,p)] + cr_{j-1}(p_{nat}(c_i,p),p))$ and $p_{nat}(c_1,p) \leq \dots \leq p_{nat}(c_{|C_v|},p)$, where $c_1,\dots,c_{|C_v|}$ are the children of v.*

Proof. Observing vertex $v \in T_2$, we subdivide its child vertices C_v into three sets, corresponding to the three cases of the natural position definition. If $p^* < \min P_{opt}(c_i)$, i.e. p^* is left of the interval, position q_i of child c_i is set to the leftmost position in the interval $q_i := \min P_{opt}(c_i)$. Symmetrically, we do the same if $p^* > \max P_{opt}(c_i)$, i.e. the ideal position is to the right of $P_{opt}(c_i)$, thus, we set $q_i := \max P_{opt}(c_i)$ rightmost within the interval. If the ideal position is a member of the optimal positions, $p^* \in P_{opt}(c_i)$, we set the position $q_i := p^*$ exactly as the ideal position. Figure 5.9 provides an example for this definition. As p^* is the same for all children of v, Claim 5.2 directly implies that $q_1 \leq \dots \leq q_{|C_v|}$ thus, $p_{nat}(c_1,p) \leq \dots \leq p_{nat}(c_{|C_v|},p)$ holds.

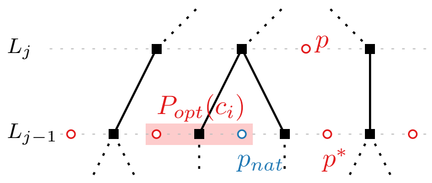


Figure 5.9: Illustrating the definition of p_{nat} shown in blue. The position p of parent v in layer j is known, thus, directly implying an ideal position p^*. The positions for which $o[c_i, q]$ is minimal form the (connected) interval $P_{opt}(c_i)$ highlighted in light red. Since for this instance p^* is strictly right of $P_{opt}(c_i)$, the natural position $p_{nat}(c_i, p)$ corresponds to the right bound of $P_{opt}(c_i)$.

We show equality of $o[v, p]$ and $\sum_{c_i \in C_v}(o[c_i, p_{nat}(c_i, p)] + cr_{j-1}(p_{nat}(c_i, p), p))$ by contradiction. Therefore assume that $o[v, p] \neq \sum_{c_i \in C_v}(o[c_i, p_{nat}(c_i, p)] + cr_{j-1}(p_{nat}(c_i, p), p))$, i.e. placing the children at their natural position does not yield an optimal solution. Then for at least one child c_i it must hold that $o[c_i, p_{nat}(c_i, p)] + cr_{j-1}(p_{nat}(c_i, p), p) > \min_{q' \in \{0,\dots n_{1|j-1}\}}(o[c_i, q'] + cr_{j-1}(q', p))$. Let $\hat{q} \in \{0, \dots, n_{1|j-1}$, $\hat{q} \neq p_{nat}(c_i, p)$ be one such placement, with a strictly lower crossing number $o[c_i, \hat{q}] + cr_{j-1}(\hat{q}, p) < o[c_i, p_{nat}(c_i, p)] + cr_{j-1}(p_{nat}(c_i, p), p)$. Due to the definition of p_{nat} and P_{opt} it must hold that $o[c_i, \hat{q}] \geq o[c_i, p_{nat}(c_i, p)]$, as $p_{nat}(c_i, p)$ is within P_{opt}.
Therefore it follows that $cr_{j-1}(\hat{q}, p) < cr_{j-1}(p_{nat}(c_i, p), p)$.
We first observe that $p^* \notin P_{opt}(c_i)$ otherwise $p_{nat}(c_i, p) = p^*$, so that $p_{nat}(c_i, p)$ would both minimize the crossings of edge (v, c_i) and o independently, i.e. $cr_{j-1}(\hat{q}, p) \geq cr_{j-1}(p_{nat}(c_i, p), p)$ which would be a contradiction.
Thus, $p_{nat}(c_i, p)$ is one of the boundaries of the interval $P_{opt}(c_i)$, i.e. $p_{nat}(c_i, p) \in \{\min P_{opt}(c_i), \max P_{opt}(c_i)\}$. W.l.o.g. assume that $p_{nat}(c_i, p) = \min P_{opt}(c_i)$. It follows that $\hat{q} < p_{nat}(c_i, p)$, otherwise $cr_{j-1}(\hat{q}, p) \geq cr_{j-1}(p_{nat}(c_i, p), p)$ which would again be a contradiction. Also this implies that $\hat{q} \notin P_{opt}(c_i)$. We will distinct between two cases next.

In the first case it holds that $\hat{q} < p^* < p_{nat}(c_i, p)$, i.e. $\hat{q}$ is further left than the ideal position. By Claim 5.1 it holds that $cr_{j-1}(p_{nat}(c_i, p), p) = p_{nat}(c_i, p) - p^*$ and $cr_{j-1}(\hat{q}, p) = p^* - \hat{q}$. Combining both equations, we get $cr_{j-1}(p_{nat}(c_i, p), p) + cr_{j-1}(\hat{q}, p) = p_{nat}(c_i, p) - \hat{q}$. By Claim 5.2 it holds that $o[c_i, \hat{q}] - o[c_i, p_{nat}(c_i, p)] \geq p_{nat}(c_i, p) - \hat{q}$. Reinserting the first equation into the second results in $o[c_i, \hat{q}] - o[c_i, p_{nat}(c_i, p)] \geq cr_{j-1}(p_{nat}(c_i, p), p) + cr_{j-1}(\hat{q}, p)$. Reformulation gives us $o[c_i, \hat{q}] - cr_{j-1}(\hat{q}, p) \geq o[c_i, p_{nat}(c_i, p)] + cr_{j-1}(p_{nat}(c_i, p), p)$. Since $cr_{j-1} \geq 0$ this implies $o[c_i, \hat{q}] + cr_{j-1}(\hat{q}, p) \geq o[c_i, p_{nat}(c_i, p)] + cr_{j-1}(p_{nat}(c_i, p), p)$ which directly contradicts the initial assumption.

For the second case it holds that $p^* \leq \hat{q} < p_{nat}(c_i, p)$, i.e. $\hat{q}$ is strictly closer to the ideal position than $p_{nat}(c_i, p)$. Using Claim 5.1 we get $cr_{j-1}(p_{nat}(c_i, p), p) - cr_{j-1}(\hat{q}, p) = p_{nat}(c_i, p) - \hat{q}$.
By Claim 5.2 it still holds that $o[c_i, \hat{q}] - o[c_i, p_{nat}(c_i, p)] \geq p_{nat}(c_i, p) - \hat{q}$. Again, reinserting the first equation into the second results in $o[c_i, \hat{q}] - o[c_i, p_{nat}(c_i, p)] \geq cr_{j-1}(p_{nat}(c_i, p), p) - cr_{j-1}(\hat{q}, p)$, which can be reformulated as $o[c_i, \hat{q}] + cr_{j-1}(\hat{q}, p) \geq o[c_i, p_{nat}(c_i, p)] + cr_{j-1}(p_{nat}(c_i, p), p)$ directly contradicting the initial assumption. □

Next we claim, that the proposed algorithm does position every vertex at its natural position for the witness.

Claim 5.4. *Considering vertex $v \in V_j(T_2)$ at position $p \in \{0, \ldots, n_{1|j}\}$ on layer j, the dynamic program does assign positions $p_{nat}(c_1, p), \ldots, p_{nat}(c_{|C_v|}, p)$ to the children $c_1, \ldots, c_{|C_v|}$ of v respectively.*

Proof. Recall, that as a tie-breaker rule, if the algorithm had multiple positions to place a child vertex c_i of v, all minimizing $o[v,p]$, the position q is chosen such that $cr_{j-1}(q,p) \geq cr_{j-1}(q',p)$ for all optimal q', i.e. the position q maximizes $cr_{j-1}(q,p)$ among all optimal placements. Observe that if $p_{nat}(c_i,p) = p^*$ then p^* is the unique placement leading to a minimum value, as both $o[c_i,q]$ is minimal at this positions and the edge (c_i, v) is crossing free, but yields at least one crossing for any position other than p^*. Thus, in this case the claim holds.

Now consider that $p_{nat}(c_i,p) \in \{\min P_{opt}(c_i), \max P_{opt}(c_i)\}$. W.l.o.g. assume that $p_{nat}(c_i,p) = \min P_{opt}(c_i)$, the case for $\max P_{opt}(c_i)$ follows symmetrically. Let position $p' \neq p_{nat}(c_i,p)$ be different position of child c_i, also leading to a minimum value of $o[v,p]$. Observe that $p' \notin P_{opt}(c_i)$ as any position in $P_{opt}(c_i)$ other than $p_{nat}(c_i,p)$ yields a larger value than $p_{nat}(c_i,p)$. For all other positions in P_{opt}, the number of crossings of edge (v,c_i) would be strictly higher due to Claim 5.1, while $o[c_i,p'] = o[c_i,p_{nat}(c_i,p)]$.
Positions to the right of $P_{opt}(c_i)$ clearly would yield even higher values, thus, $p' < p_{nat}(c_i,p)]$. Further, positions $p' < p^*$ also clearly yield non-minimal values, therefore $p^* \leq p' < p_{nat}(c_i,p)]$ must hold. Using Claim 5.2 we get $o[c_i,p'] - o[c_i,p_{nat}(c_i,p)] \geq p' - p_{nat}(c_i,p)$. Thus, increasing the distance between p' and $p_{nat}(c_i,p)$ increases $o[c_i,p']$ by at least one, while, due to Claim 5.1, decreasing $cr_{j-1}(p',p)$ by exactly one. Therefore the algorithm chooses position $p_{nat}(c_i,p)$ as $cr_{j-1}(p_{nat}(c_i,p),p) > cr_{j-1}(p',p)$ for all positions yielding minimal crossings. □

Using these intermediate results enables us to prove the key lemma of this section.

Lemma 5.2. *The drawing constructed by the dynamic program embeds T_2 according to ε_2.*

Proof. We prove this by induction over layer j, beginning with layer $j = L(r_2)$, showing that the dynamic program assignes positions to the vertices of T_2 in layer j, so that they satisfy ε_2.

For the base case $j = L(r_2)$ there is only one vertex of T_2 on layer j, namely r_2. Therefore the proposition trivially holds.

For the induction case $0 < j < L(r_2)$, we assume the proposition is true for all layers $> j$ by induction. Let $v_1, \ldots v_{n_{2|j+1}}$ denote the vertices of the consecutive layer $j+1$. Using Claims 5.3 and 5.4, considering vertex v_i in layer $j+1$, the children C_{v_i} of v_i on layer j are assigned to positions respecting the order $\prec_j^2$. As v_i and C_{v_i} describe a star, its edges are crossing free. Thus, all edges sharing an endpoint are crossing free.
However it remains to show, that no pair of crossing edges between layers j and $j+1$ exists. which do not have any common endpoint. Let v_i and $v_{i'}$

be two vertices of layer $j+1$, w.l.o.g. $i < i'$. By the induction hypothesis v_i and $v_{i'}$ are ordered according to $\prec^2_{j+1}$. Thus, $p_i \leq p_{i'}$ where p_i and $p_{i'}$ denote the positions of v_i and $v_{i'}$ respectively. By Claim 5.1 it holds, that the ideal positions of the children of v_i and $v_{i'}$ are also ordered accordingly, i.e. $p^*_i \leq p^*_{i'}$. Additionally, by Claim 5.2 the min and max values of P_{opt} of $v_1, \ldots v_{n_{2|j+1}}$ are ordered correspondingly. Thus, by Claim 5.4 and the definition of p_{nat} it holds that $q_i \leq q_{i'}$ where q_i denotes the position of a child of v_i and $q_{i'}$ the position of a child of $v_{i'}$ for any pair of child vertices in layer j. Clearly this also implies, that all edges between layers j and $j+1$ are crossing free, thus, ε_2 is satisfied in layer j. □

As each layer satisfies ε_2 the full drawing respects ε_2. Finally, to conclude the correctness prove, it remains to show the optimality of the provided algorithm. To do so, we state the following.

Lemma 5.3. *The constructed drawing minimizes in the number of edge crossings.*

Proof. Again we use induction to prove this property, specifically we induce over the layers j with $j = 2$ as the base case. For layer j we show that for vertex v at position p, $o[v,p]$ is exactly the minimum number of crossings between edges of T_1 and edges of the tree T_v which is the subtree of T_2 rooted at vertex v. Since the algorithm assumes a fixed placement of T_1 according to ε_1 and Lemma 5.3 showed, that no two edges of T_2 cross in the constructed drawing, $o[v,p]$ denotes the total number of crossings of the drawing induced by $T_1 \cup T_v$.

For $j = 2$ the proposition clearly holds as $o[v,p]$ is the sum of crossings between incident edges to v and T_1.

Now consider $j \geq 3$. For the children $c_1, \ldots c_{|C_v|}$ of v we know that the algorithm assigns positions $p_{nat}(c_1,p), \ldots, p_{nat}(c_{|C_v|},p)$, respectively, as shown in Claim 5.4. By the induction hypothesis it holds that for each $c_i \in C_v$ the drawing of T_{c_i}, with c_i at position $p_{nat}(c_i,p)$, with minimum number of crossings has exactly $o[c_i, p_{nat}(c_i,p)]$ crossings. Additionally the crossings $cr_{j-1}(p, p_{nat}(c_i,p))$ are added, corresponding to the number of crossings of edges between layers $j-1$ and j. By the formulation of the dynamic program, the positions of $c_1, \ldots c_{|C_v|}$ are assigned such that the total number of crossings induced by T_v given in $o[v,p]$ is minimal. □

Combining the Lemmas 5.1 to 5.3, we have proven the final theorem.

Theorem 5.4. *Given a forest F of n vertices, which consists of exactly two trees $F = \{T_1, T_2\}$ with layered upward-planar drawings ε_1 and ε_2 respectively, where all leaves in layer 1. A total order $\prec_1$ of all leaves is given. The crossing minimal drawing of F, preserving $\varepsilon_1, \varepsilon_2$ and $\prec_1$ can be computed in $\mathcal{O}(n^3)$ asymptotic time.*

5.2 Generalizations of CMUT for two trees

Additionally we remark that this algorithmic solution can be extended to solve several other general problems.

The first generalization adds further requirements to the drawing. Specifically the CMUT problem is augmented with additional (partial) orders for layers $2, \ldots, \ell$ denoted as $\prec_2, \ldots, \prec_\ell$, which have to be satisfied in the resulting drawing. This variation can solve cases where layers other than the leaves should also be sorted by some criteria, to encode additional information in the visualization. Clearly the original CMUT problem is included in this broader definition, also it does not suffice to use the solution provided by the proof of Theorem 5.4 and replace the vertices in a second pass, to fulfill $\prec_2, \ldots, \prec_\ell$, as this will generally increase the total number of crossings.

Remark 5.1. *Given a forest F of n vertices, which consists of exactly two trees $F = \{T_1, T_2\}$ with layered upward-planar drawings ε_1 and ε_2 respectively, where all leaves in layer 1. Additionally a total order $\prec_1$ of all leaves and (partial) orders $\prec_2, \ldots, \prec_\ell$ for all other layers are given. The crossing minimal drawing of F, preserving $\varepsilon_1, \varepsilon_2$ and $\prec_1, \ldots, \prec_\ell$ can be computed in $\mathcal{O}(n^3)$ asymptotic time.*

Proof. The algorithm as provided in the proof of Theorem 5.4 can be applied, however for any $v \in T_2$ in layer j, for any position p in layer j, contradicting $\prec_j$, we set $o[v,p] := \infty$ instead of the recursive definition. This can be done during the bottom-up process. This ensures that such positions are chosen only when no position exists satisfying the given (partial) orders. Thus, a final check whether the resulting optimal number of crossings is lower than ∞ states, whether a valid solution exists. The proof of this algorithm variation follows analogously to the proof of Theorem 5.4. □

So far, the problem required that all leaves to be placed in layer 1, as well as a total order of layer 1. In the following we will relax this requirement to further extend the provided results. However note that the restriction of T_2 to ε_2 has to be dropped. Also the crossings which are minimized will be specified by the crossings between T_1 and T_2 only, i.e. crossings between a pair of edges where one edge belongs to T_1 and one edge to T_2. Moreover we remark, that minimizing the total number of crossings for this setting is NP-complete, as we can easily reduce the NP-complete problem of minimizing the number of crossings for stars on 2 layers [95] to it. We do this by setting T_1 to be a tree without vertices. T_2 is a tree with two layers, the leaves are all in layer 1, those correspond to the leave vertices of the NP-complete problem. The leaves are fixed in position by providing a total order of $\prec_1$. No order is provided for layer 2. Therefore the vertices corresponding to the center points of stars can be freely rearranged. Layer 3 consists of only r_2. Clearly all incident edges to $_2$ are crossing free, thus rearranging the center points in layer 2 minimizes the total number of crossings.

Remark 5.2. *Given a forest F of n vertices, which consists of exactly two trees $F = \{T_1, T_2\}$ and a layered planar upward drawings ε of T_1. The vertices of T_2 are assigned to layers. Additionally (partial) orders $\prec_1, \ldots, \prec_\ell$ for all layers $1, \ldots, \ell$ are given. The drawing of F with minimal number of crossings, preserving ε and $\prec_1, \ldots, \prec_\ell$ can be computed in $\mathcal{O}(n^3)$ asymptotic time.*

Proof. Note that in contrast to Theorem 5.4, the leaves of T_2 are not necessarily assigned to layer 1, nor must there be a total order for layer 1 as part of the input. However, as the drawing of T_1 is fixed, the algorithm as in Remark 5.1 can still be applied. Although, since the leaves in layer 1 are no longer fixed by a complete order, the positions of the leaves have to be computed. Setting $o[v, p] = 0$ for every position p satisfying $\prec_1$ and $o[v, p] = \infty$ for every position p not satisfying $\prec_1$ allows us to use the general definition for layers ≥ 2 instead of > 2. The proofs provided for Theorem 5.4 and Remark 5.1 still hold, except for the proof of planarity of T_2 in the resulting drawing. The proofs of correctness and running time remain the same.

Note that if all leaves are on layer 1 and $\prec_1$ is a total order, the problem is equivalent to the CMUT problem. $\square$

We lastly remark, that this approach can also be extended to solve the problem for a set of one tree and one planar graph.

Remark 5.3. *Given a planar graph G with a layered upward-planar drawing ε_G and a tree T, with a layered upward-planar drawing ε_T where all leaves of T are in layer 1. Further a total order $\prec_1$ as well as partial orders $\prec_2, \cdots \prec_k$ for the vertices in layers $1, \ldots, k$ are given. The crossing minimal drawing of F preserving ε_G, ε_T and $\prec_1, \cdots \prec_k$ can be computed in $\mathcal{O}(n^3)$ asymptotic time, where n denotes the total amount of vertices.*

Proof. Only small adaptions of the dynamic programming algorithm are necessary. G will be considered fixed according to ε_G, similar as T_1 in the original algorithm. T will then be drawn according to the optimization algorithm, resembling T_2 in the original algorithm. All possible positions are considered to calculate the optimal partial solutions and the optimal placement of the root can be found in linear time. The crossings occurring between T and G are independent of the structure of G, as the original dynamic program does not utilize the restriction that T_1 is a tree. Correctness and time complexity follow analogously. $\square$

Chapter 6

Limited Height

In this chapter, we relax the number of trees. While it is known that for an unbounded number of trees this problem is NP-complete, even for two layers only, when taking the number of layers as part of the input we propose an FPL-algorithm if the number of layers is limited to two and an XP-algorithm if the number of layers is limited to three.
Before we delve into the results, we make a quick observation. It is impossible to solve this problem by finding pairwise optimal drawings using Theorem 5.4 for two trees at a time and then combining the drawings into one. This is clear from the NP-hardness of the problem even for two layers (but an arbitrary number of trees).

6.1 CMUT for Two Layers

Theorem 6.1. *Given a forest F of k directed rooted and layered trees with n vertices in total, where all roots are on the second layer while all the leaves are on the first layer in a fixed total order $\prec_1$. A minimum crossing drawing of F preserving $\prec_1$ can be computed in $\mathcal{O}(2^k k^2 + nk)$ time. The problem is fixed parameter linear.*

Remark that the trees correspond to stars with center points in layer 2 due to the layer restriction. We will still refer to them as trees to keep it consistent with the rest of this chapter.

We will present an algorithm which determines a total order $\overset{!}{\prec}_2$ of all vertices of the second layer, such that the number of total crossings is minimal. The algorithm will solve the problem instance by constructing an equivalent shortest path problem on a k-dimensional cube graph. To describe the algorithm, some further notation is helpful. Let σ denote an arbitrary but fixed order of the vertices of layer two, then the position of a vertex $v \in V_2(F)$ relative to any given tree $i \in \{1, \ldots, k\}$ will be denoted as $p^i_{v,\sigma}$. Since all vertices in layer 2 are roots $p^i_{v,\sigma} \in \{0, 1\}$, where 0 corresponds to the case where v is left of r_i and 1 corresponds to the case where v is right of r_i according to σ. Thus, the (total) position of a vertex in the drawing, resembling σ can be uniquely described by

its positions relative to all other $k-1$ roots.
We observe, that in a drawing using σ as the order of the roots, every crossing can be charged to exactly two vertices of $V_2(F)$ as a crossing between two edges e_1 and e_2 will be charged to the two roots incident to e_1 and e_2.

Claim 6.1. *The crossings of a root r_i depend solely on $p^j_{r_i,\sigma}, \forall j \in \{1,\dots k\} \setminus \{i\}$.*

Proof. Consider the individual drawing of a tree T_j, along with all leaves of layers 1. The total order of the leaves is known. Now when root r_i with all incident edges is inserted at position $p^j_{r_i,\sigma}$, all resulting crossings are charged to root r_i. We denote the resulting number of crossings $cr^j_{r_i,p}$, where $p = p^j_{v,\sigma}$. Note that $cr^j_{r_i,p}$ corresponds to all crossings between tree j and tree i. The total number of crossings charged to r_i is denoted as $cr_{r_i,\sigma} = \sum_{j\in\{1,\dots,k\}\setminus\{i\}} cr^j_{r_i,p}$. Thus, the total number of crossings of the drawing obtained by σ is $cr_\sigma = \frac{\sum_{i\in\{1,\dots,k\}} cr_{r_i,\sigma}}{2}$, as every crossing is charged twice, once for each incident root. □

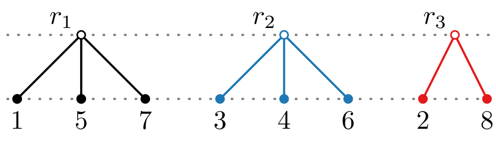
Further we observe that these charged crossings can be efficiently computed.

Claim 6.2. *The values $cr^j_{r_i,p}$ for all combinations of $i, j \in \{1,\dots,k\}, i \neq j$ and $p \in \{0,1\}$ can be computed in total time $\mathcal{O}(k \cdot n)$.*

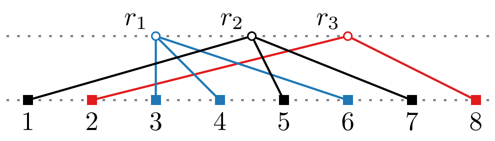
Proof. There are $\mathcal{O}(k)$ trees in total. Fixing one tree and iterating through all leaves lets us calculate $cr^j_{r_i,p}$ for all j edge by edge, as any edge which is not part of T_i belongs to one tree j, for which the two values $cr^j_{r_i,0}$ and $cr^j_{r_i,1}$ have do be determined. Depending on the position of the leaves in the total order of layer 1, the number of crossings follows directly. Thus, calculating all $cr^j_{r_i,p}$ for a fixed i can be done in $\mathcal{O}(n)$. As there are k trees, we get $\mathcal{O}(k \cdot n)$ in total. □

We now reduce the problem to a shortest-path problem utilizing our observations. To do so, we construct a weighted directed acyclic st-graph H, see Figure 6.1 for a full example. The st-paths of H represent all permutations of the roots. Furthermore H is constructed such that an st-path denoted as π which represents a total order σ of layer 2 has a summed weight of twice cr_σ, i.e. twice the number of crossings the drawing implementing σ has.

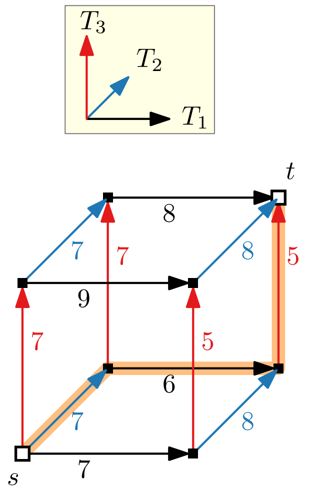
Construction of H: Let H be the k-dimensional cube graph whose edges are directed from the corner $(0,\dots,0)$ to the corner $(1,\dots,1)$. Specifically H has the node set $\{(x_1,\dots,x_k) \mid x_1 \in \{0,1\},\dots,x_k \in \{0,1\}\}$. The edge set consists of all edges $\{(x_1,\dots,x_k),(y_1,\dots,y_k)\}$ where $x_j + 1 = y_j$ holds for exactly one $j \in \{0,\dots,k\}$, while $x_i = y_i$ holds otherwise. Thus, any edge corresponds to the traversal from one cube corner to another along an outer edge (as there are no diagonals). Observe that the corner node $(0,\dots,0)$ is the unique source, while $(1,\dots,1)$ is the unique sink. We denote $(0,\dots,0)$ as s and $(1,\dots,1)$ as t.
For an edge of the edge set, let this edge be from node $(x_1,\dots,x_k)$ to $(y_1,\dots,y_k)$, where $x_i + 1 = y_i$, the edge be weighted by $\sum_{j\in\{1,\dots,k\}\setminus\{i\}} cr^j_{r_i,x_j}$. Note that x_i must be 0 while $y_i = 1$. Traversing along this edge in an st-path represents placing the root of T_i left of the root r_j of T_j if $x_j = 0$ and right of r_j if $x_j = 1$. The weight of the edge corresponds to the number of crossings charged to r_i when placing it at this relative position.



(a) Three trees (stars) are given, with the roots (centerpoints) on layer 2. The trees are shown side by side, however the order of the leaves is fixed as indicated by the labels.



(b) Crossing minimal drawing of the forest preserving the order of the leaf layer. The roots are placed according to the shortest weighted st-path of H, highlighted in orange in (c).



(c) Cube graph H whose st-paths represent all permutations of the roots. The st-path highlighted in orange is the shortest weighted st-path with total weight 18, corresponding to a drawing with 9 crossings.

Figure 6.1: Reducing the problem of finding a crossing minimal drawing on two layers to a shortest-path problem in a weighted k-dimensional cube.

Traversing an st-path informally corresponds to incrementally placing the roots strictly from left to right. We start at s with no placed roots, and when advancing in the dimension i representing tree T_i we place the root of T_i until, when reaching t all roots have been placed, thus, we have determined a total order of layer 2. Clearly, any st-path π of H is of length k. As the weight of an edge corresponds to the number of crossing the root induces at this position, finding a crossing-minimal total order of the roots is equivalent to finding a shortest st-path in H. Note that, since we have a k-dimensional cube graph, any st-path in H of minimal length traversed exactly one edge 'in' each dimension. However all permutations of the order in which the different dimensions are traversed are represented by st-paths. Thus, there is a bijection between the set of st-paths and the set of valid total orders σ.
We obtain an FPL-algorithm by construction of H and applying a shortest st-path algorithm on H.

Claim 6.3. *The algorithm has a time complexity of $\mathcal{O}(2^k k^2 + kn)$.*

Proof. Since H is a k-dimensional cube it consists of 2^k nodes. As the in degree of any node is at most k, one incoming edge corresponding to each dimension, the total number of edges is $E(H) < 2^k k$. For every edge the weight has to be computed by summing $k-1$ values of $cr^i_{v,p}$, which can be pre-computed in $\mathcal{O}(n \cdot k)$ time, as shown in 6.2. Thus, constructing H including the edge weights can be done in $\mathcal{O}(2^k k^2 + kn)$.

Finding the shortest weighted st-path in H can be done in $\mathcal{O}(2^k k)$ time by using topological ordering[35]. The optimal drawing can be constructed in linear time once the shortest weighted st-path is known. Thus, the construction of H dominates the running time yielding a total asymptotical running time of $\mathcal{O}(2^k k^2 + kn)$. Therefore the problem is in FPL. □

6.2 CMUT for Three Layers

Increasing the number of layers to three we obtain the following result:

Theorem 6.2. *Given a forest F of k directed rooted and layered trees with corresponding embeddings, where all roots are on the third layer while all leaves are on the first layer in a fixed order $\prec_1$, a minimum crossing drawing of F preserving the embeddings and $\prec_1$ can be computed in $\mathcal{O}(k^2 n^k)$ time, where n is the total number of vertices.*

Observe that in contrast to Theorem 6.1, there are linearly many vertices in layer 2, which are restricted by the embeddings of the individual trees. Further an order of the roots in layer 3 has to be determined. However, we can utilize the same basic idea as in Theorem 6.1 to reduce the problem to a shortest path problem, which provides an XP-algorithm. However significant changes have to be made to the approach to accommodate for the additional layer. Thus, in the following, we will describe the algorithm in detail. Particularly H will no longer describe a cube graph, but a general k-dimensional grid graph, where the st-paths correspond to permutations of the second layer.

First, we observe that the third layer consists of only the roots, thus, of exactly k vertices. Therefore $k!$ permutations for the total order of layer 3 exist. The algorithm will handle all of these $k!$ cases individually and select the case with the lowest number of crossings as optimal solution. Thus, in the following we will consider not only the leaves as having a fixed order $\prec_1 = \overset{!}{\prec}_1$, but the roots as well. Let $\overset{!}{\prec}_3$ denote the total order of the third layer. Thus, it suffices to find a total order $\overset{!}{\prec}_2$ of all vertices of the second layer, respecting the planarity of all trees individually, which can be ensured by preserving the given embeddings. We denote the additional partial orders, which are directly implied by the order of the leaves of all trees as $\prec_2^1, \ldots, \prec_2^k$ for trees $T_1, \ldots, T_2$ respectively.

For comprehensibility of the algorithm we introduce some further notation. Let σ denote an arbitrary but fixed order of the vertices of layer 2 consistent with $\prec_2^1, \ldots, \prec_2^k$. The position of a vertex $v \in V_2(F)$ relative to any given tree $i \in \{1, \ldots, k\}$ will be denoted as $p_{v,\sigma}^i$. We define $p_{v,\sigma}^i$ as the position in left-to-right manner within the vertex set $V_2(T_i) \cup \{v\}$ according to σ and starting at 0, i.e. the number of vertices of T_i of layer 2 which are left of v. Note that in contrast to Theorem 6.1 there can be linearly many positions. The position of a vertex in the drawing implied by σ can be uniquely described by its positions relative to all other $k-1$ trees. Similar in Theorem 6.1, for a drawing implied by σ, every crossing can be charged to exactly two vertices of $V_2(F)$, by charging a crossing between two edges e_1 and e_2 to the two out of four endpoints, which are on layer two. Note that in contrast to the CMUT problem on two layers, crossings appearing both between layers 1 and 2 as well as crossings between layers 2 and

3 are charged to the vertices in layers 2. Their order only is determined by the algorithm, all other vertices are considered fixed. This means, that we construct a H for every permutation of the roots.

Claim 6.4. *The crossings of a vertex $v \in T_i$ of layer two depend solely on $p^j_{v,\sigma}$ for all $j \in \{1, \dots k\} \setminus \{i\}$.*

Proof. Consider the individual drawing of T_j, along with all vertices of layers 1 and 3, for which the order is fixed. Now when vertex v with all incident edges (describing a star, with exactly one edge between layers 2 and 3 and all other edges between layers 1 and 2) is inserted at position $p^j_{v,\sigma}$, all resulting crossings are charged to vertex v. We denote the resulting number of crossings by $cr^j_{v,p}$, where $p = p^j_{v,\sigma}$. Note that $cr^j_{v,p}$ corresponds to all crossings between tree T_j and incident edges of v. The total number of crossings charged to v is denoted as $cr_{v,\sigma} = \sum_{j\in\{1,\dots,k\}\setminus\{j\}} cr^j_{v,p}$. Thus, the total number of crossings of the complete drawing, when using σ, is $cr_\sigma = \frac{\sum_{v\in V_2(\mathcal{F})} cr_{v,\sigma}}{2}$. □

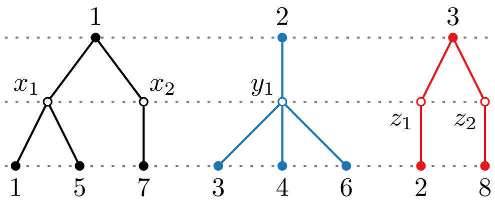
Again, we show that these crossings can be pre-computed, although not as efficiently as in Theorem 6.1.

Claim 6.5. *The values $cr^j_{v,p}$ for all combinations of $j \in \{1, \dots, k\}$, $v \in V_2(\mathcal{F}) \setminus V_2(T_j)$ and $p \in \{0, \dots, |V_2(T_j)|\}$ can be computed in total time $\mathcal{O}(n^2)$.*

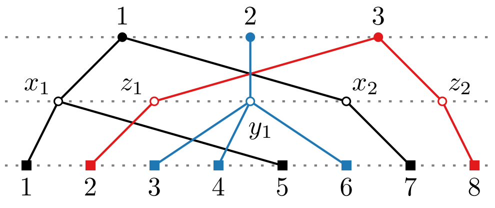
Proof. Observe that every vertex $v \in V_2(\mathcal{F})$ with its incident edges and adjacent neighbors forms a star, which we denote as S_v. S_v has one edge between layers 2 and 3 and all other edges between layers 1 and 2. As F is a forest, all stars can be constructed and saved in $\mathcal{O}(n)$ time in total. Now we consider vertex $v \in V_2(T_i)$ and tree T_j, $i, j \in \{1, \dots, k\}, i \neq j$ to be fixed. We compute $cr^j_{v,0}$ by inserting v as the leftmost vertex in the second layer and considering all pairs of edges of S_v and T_j, returning, whether they cross. Now we increment $p = 1, \dots, |V_2(T_i)|$ and update the number of crossings accordingly. Note that it suffices to update crossings occurring between S_v and the star induced by the p-th vertex of $V_2(T_i)$ and its neighborhood. We observe, that overall any pair of crossing edges is considered at most four times, thus, the total running time is $\mathcal{O}(n^2)$. □

We now reduce the problem to a shortest-path problem. To do so we construct a weighted directed acyclic st-graph H, for a running example see Figure 6.2). Note that H represents the placements of the inner vertices of $\mathcal{F}$ instead of the roots, as in 6.1. Naturally this increases the structural size of H significantly. The set of all st-paths of H represent exactly all total orders of $V_2(\mathcal{F})$ respecting the partial orders $\prec^1_2, \dots, \prec^k_2$. Furthermore, an st-path denoted as π which represents a total order σ of layer 2 has a summed edge weight of twice cr_σ, i.e. twice the number of crossings the drawing defined by σ yields.

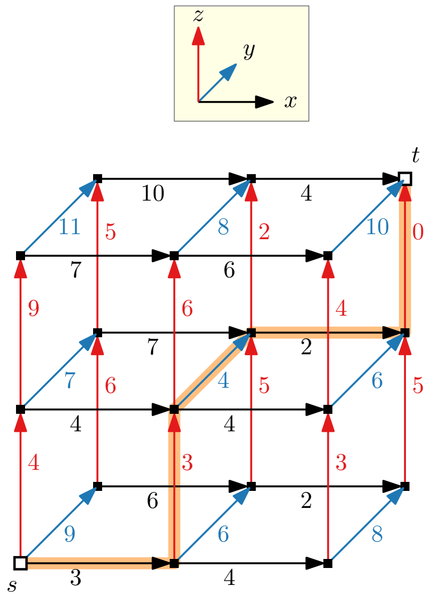
Construction of H. Let H be the k-dimensional grid graph of side lengths $|V_2(T_1)| \times \dots \times |V_2(T_k)|$ whose edges are directed from the corner $(0, \dots, 0)$ to the corner $(|V_2(T_1)|, \dots, |V_2(T_k)|)$. Specifically H consists of the node set $\{(x_1, \dots, x_k) \mid x_1 \in \{0, \dots, |V_2(T_1)|\}, \dots, x_k \in \{0, \dots, |V_2(T_k)|\}\}$. The edge set consists of all edges $\{(x_1, \dots, x_k), (y_1, \dots, y_k)\}$ for which $x_j + 1 = y_j$ holds for



(a) Three trees on three layers are given as input. The trees are shown side by side, however the total order of the vertices on the first layer is fixed (indicated by the indices of the leaves), while all permutations of the third layer are tested. For this example we consider the permutation as indicated by the indices of the roots.



(b) Crossing minimal drawing of the forest, preserving the orders of layer 1. The order of the vertices in layer 2 corresponds to the shortest weighted st-path of H, highlighted in orange in (c).



(c) Grid graph H whose st-paths describe the total orders of the vertices on the second layer preserving the given embeddings. The st-path highlighted in orange is the shortest weighted st-path with total weight 12, corresponding to 6 crossings in the optimal drawing.

Figure 6.2: Reducing the problem of finding a crossing minimal drawing on three layers to a shortest-path problem in a weighted k-dimensional grid graph.

exactly one $j \in \{0, \ldots, k\}$, while $x_i = y_i$ holds otherwise. Thus, any edge informally corresponds to an intermediate step at placing the vertices of layer 2 from left to right. Specifically it correponds to the placement of the $x_j + 1$-th vertex of T_j, let this vertex be denoted as v. The node corresponds placing v such that x_1 vertices of T_1 are placed left of v, x_2 vertices of T_2 left of v and so forth. A node corresponds to an intermediate state having saved for any tree T_i how many vertices of T_i already have been placed, which is encoded in x_i.
Observe that the corner node $(0, \ldots, 0)$ is the unique source while corner node $(|V_2(T_1)|, \ldots, |V_2(T_k)|)$ is the unique sink of H. We denote $(0, \ldots, 0)$ as s and $(|V_2(T_1)|, \ldots, |V_2(T_k)|)$ as t.
Let e be an edge of the edge set, where e is an edge from node $(x_1, \ldots, x_k)$ to $(y_1, \ldots, y_k)$, with $x_j + 1 = y_j$. Then e is weighted by $\sum_{i \in \{1,\ldots,k\} \setminus \{j\}} cr^i_{v,x_i}$. Let the $x_j + 1$-th vertes of T_j be denoted as v. Thus, the weight of e corresponds to the number of crossings charged to v when placing it such that exactly x_i vertices are left of v for every tree $T_i \neq T_j$.
Again any st-path informally corresponds to incrementally placing the vertices of layer 2 strictly from left to right starting at s with no placed vertices, and when advancing in the dimension i representing tree T_i, we place the next vertex of T_i in left to right order, according to its planar embedding. When reaching t

all vertices (of layer two) of all trees have been placed, thus we determined a total order of the vertices of layer 2.
Clearly, any st-path π of H is of length n_2, where n_2 denotes the number of vertices in layer 2. As the weight of an edge corresponds to the number of crossing the vertex induces at this placement, finding the crossing-minimal total order of layer two is equivalent to finding a shortest st-path in H. Note that, since we have a k-dimensional grid graph, any st-path in H traverses $|V_2(T_1)|$ edges 'in' the first dimension, $|V_2(T_2)|$ edges 'in' the second dimension, and so forth. The vertices corresponding to the trees can be interleaved arbitrarily, thus, there is a bijection between the set of st-paths and the set of valid total orders σ.
The total order is naturally consistent with the given planar embeddings, as the vertices of every tree can be only placed in a fixed order, described by $\prec_2^1, \dots, \prec_2^k$. With this we obtain an XP-algorithm finding an optimal solution.

Claim 6.6. *The algorithm has a time complexity of $\mathcal{O}(k^2 n^k)$.*

Proof. Grid graph H is constructed once for each permutation of the roots, thus H is constructed $\mathcal{O}(n!)$ times.

For the running time of the shortest path algorithm, we need to determine the number of edges in H, which is

$$\begin{aligned} E(H) &= \sum_{i=1}^{k} |V_2(T_i)| \prod_{j \in \{1,\dots,k\} \setminus \{i\}} (|V_2(T_j)| + 1) \\ &\leq k \prod_{i=1}^{k} (|V_2(T_i)| + 1) \\ &\leq k \left(\frac{n}{k}\right)^k . \end{aligned}$$

The edge weights can be computed by summing $k-1$ values of $cr^i_{v,p}$. We can be pre-compute all values of $cr^i_{v,p}$ in $\mathcal{O}(n^2)$ time, as shown in Claim 6.5. Thus, constructing H including edge weights can be done in $\mathcal{O}(k^2(\frac{n}{k})^k)$. A shortest st-path of H can be found in $\mathcal{O}(k(\frac{n}{k})^k)$ time by using topological ordering [35]. The crossing minimal drawing of $\mathcal{F}$ is obtained by constructing and solving H for all $(k!)$ permutations and choosing the crossing-minimal. With a known total order of layer 2, the drawing can be constructed in linear time. Thus, the total time complexity is $\mathcal{O}(k!k^2(\frac{n}{k})^k) \subseteq \mathcal{O}(k^2 n^k)$. □

As with the CMUT problem for two trees, the problem can be generalized to some extend while still obtaining the same results. We do so by making small adjustments to the provided algorithms.

First, we accept (partial) orders for layers 2 and 3 as part of the input, which have to be preserved in the final drawing. Simultaneously the restriction on leaf vertices being in layer 1 and roots being in layer 3 can be dropped. Thus, we get the slightly stronger result:

Remark 6.1. *Given a forest F of k directed rooted and layered trees with corresponding embeddings, where the order of layer 1 is fixed and partial orders* *for layers 2 and 3 are given as $\prec_1, \prec_2$ and $\prec_3$, a minimum crossing drawing of F preserving the embeddings and given orders can be computed in $\mathcal{O}(k^2 n^k)$ time, where n is the total number of vertices.*

Proof. We have to make small adjustments to the algorithm provided in the proof of Theorem 6.2 to accommodate for the additional requirements. Relaxing the layering of leaves and roots does not effectively affect the algorithm, as leaves in layer two can be handled exactly like inner vertices in layer two. They have no incoming edges, thus, less crossings are possible, however this does not affect the crossing calculation for any fixed position. Observe that $\prec_2^i$ for T_i still is a complete order regarding the vertices of T_i in layer 2.The partial order $\prec_2^i$ directly follows from a planar embedding, implying also the order of leaves in layer 2. Leaves in layer three coincide with roots and are thus isolated vertices which are trivially handled. Roots in layer two are also ordered by the shortest path problem on H, computing the optimal order of all vertices of layer 2. This reduces the number of vertices in layer 3, which in turn reduces the number of permutations which have to be considered, however the permutations are clearly still bounded by $k!$. Roots in layer 1 coincide with leaves and can be trivially handled.
Regarding the given (partial) orders $\prec_2$ and $\prec_3$, the (partial) order $\prec_3$ can easily preserved by discarding any permutation of the roots of layer 3 contradicting $\prec_3$, which can be verified in time $\mathcal{O}(k)$.
To preserve (partial) order $\prec_2$, the weights of edges of H have to be adapted. If an edge, corresponding to a relative placement of some vertex v of layer 2 contradicts $\prec_2$, the edge weight is set to ∞. Clearly this implies the crossings of an order σ are $cr_\sigma = \infty$ exactly if $\overset{!}{\prec}_2$ corresponding to σ contradicts $\prec_2$. Therefore it can easily verified if a valid solution exists. As the compatibility of an edge in H with $\prec_2$ can be checked in linear time, its running time is dominated by that of the edge weight computation. Thus, correctness and running time follow analogously to the proof of Theorem 6.2. □

As the algorithm provided relied heavily on layer 1 and 3 being fixed, but not on the specific graph structure, $\mathcal{F}$ must not be a forest for the provided algorithm to be applicable. However the number of vertices in layer 3 must be small to bound the running time.

Remark 6.2. *The result also holds for k upward-planar graphs if there are $\mathcal{O}(k)$ vertices on layer 3.*

The correctness of this claim directly follows from the proof of Theorem 6.1, as the tree properties are not used in the proof.

Part III

Ortho-Radial Morphing

Ortho-Radial Morphing

Orthogonal drawings are among the best studied drawing models in Graph Drawing research [41, 85, 127], with frequent use in practical applications, such as diagramming in software engineering [56] or VLSI design [62]. A drawing is *orthogonal* if edge segments are either drawn horizontally or vertically, with all vertices and edges being placed on a rectangular grid. One extension of this drawing model, which received growing attention recently, are the *ortho-radial drawings* [6, 70, 97].
An ortho-radial drawing is a graph drawing where all vertices and edges are drawn on a grid consisting of concentric circles and spokes, intersecting the center point. Naturally all edge intersections appear in orthogonal or co-linear fashion. Thus, informally, one can think of a ortho-radial drawing as a translation of orthogonal drawings into polar coordinates. To this end, an ortho-radial drawing can be interpreted as an orthogonal drawing on a cylinder, which is projected back onto the plane by connecting two opposing sides of the grid [71]; see Figure 7.1. This results in a radial grid, formed by concentric circles and spokes emanating from a center point, where no grid point lies within the innermost circle. Note that the extension of the plane grid allows to reduce the number of total bends [96], which is a frequently applied quality criterion, especially for orthogonal graph drawings [106].

Ortho-radial drawings find their main applications in metro maps [5, 114], displaying transportation networks to the public, while also being used for hypergraph and set visualization [58, 80]. Metro maps are traditionally drawn manually, see Figure 7.2 for a hand-crafted visualization of the New York metro map; however recently the automatic creation of such metro maps has been studied [7, 26, 27, 98, 101, 136].

In our work, we consider the related research area of morphings, that has been extensively studied for orthogonal drawings but has not previously been investigated for ortho-radial drawings.
A *morph* of a drawing is a transformation of one drawing of a graph into another drawing of the same graph, by defining a series of continual intermediate drawings. This is necessary when a visualization is not only automatically generated, but should be interactive and dynamic, allowing for (partial) rotation, translation and resizing.
In Chapter 8 we will propose different morphing strategies and study their properties. We observe that for many practical applications, octo-linear drawings are chosen over orthogonal drawings. Thus, analogously, we will extend the

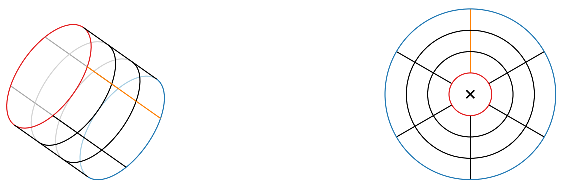


Figure 7.1: A 4 by 6 grid projected to (a) a cylinder,(b) the plane, where it is drawn as an ortho-radial grid.

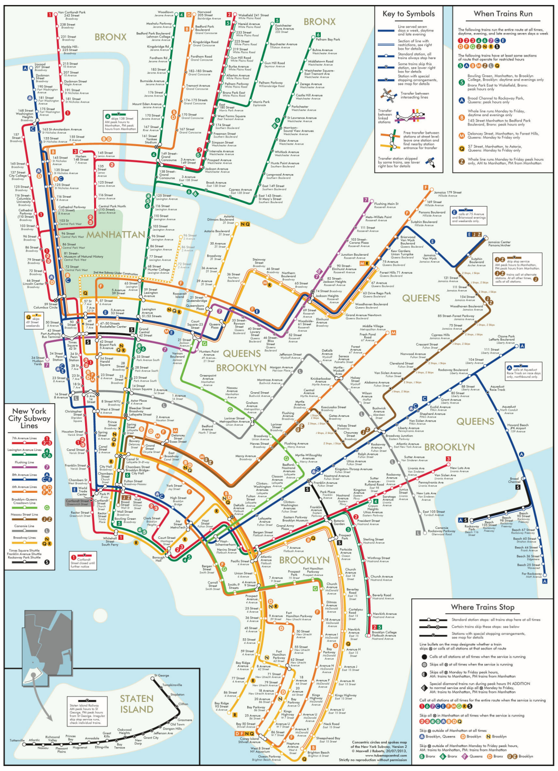


Figure 7.2: An ortho-radial drawing of the New York metro by Maxwell J. Roberts, taken from [113].

ortho-radial model to allow for 45° angle between crossing segments at the intersection, which is achieved by allowing curve segments on logarithmic spirals. In Section 7.4 we compare the performance of the different proposed strategies. To do so, we perform statistical analysis on several quality metrics on provided benchmarks and randomized data.[1]
Lastly in Chapter 8 we propose a hybrid visualization model utilizing these techniques.

[1]The resulting visual animations and the source code are provided at `https://github.com/TheAnonymousSubmitter/MorphingOrthoradialDrawings`.

Chapter 8

Morphing Strategies

7.1 Definitions

Given a graph $G = (V, E)$, for a time interval $t \in [0, 1]$ a *morph* between two drawings Γ_0 and Γ_1 of the same graph G is a series of drawings, where for any point t in the time interval a drawing Γ_t is defined. A drawing Γ of G is denoted as *ortho-radial*, if all edge segments are drawn on the grid consisting of concentric circles and spokes emanating from the center point. All vertices must be placed on grid points. Note that, to enable completely continuous morphs, we consider drawings on ortho-radial grids with infinite resolution. If the placement of vertices is continuous, the morph is called *smooth*.

Analogously to octilinear drawings [41, 85, 127], which extend the orthogonal drawing model to allow for more slopes, by introducing grid lines which cross the other grid lines at 45° angle, i.e. by adding diagonals to the grid, we introduce the model of *octo-radial* drawings. Recall, that the logarithmic spiral, also known as the equiangular spiral, is of constant slope angle. A drawing is octo-radial, if all edge segments are drawn on the grid consisting of concentric circles, spokes emanating from the center point and logarithmic spirals, as defined by $r = ae^{k \cdot \phi}$, where r is the radius, e is Euler's number, ϕ is the angle in polar coordinates, $a > 0$ is a real constant and k is $\in \{-1, 1\}$. Thus all intersections between the spiral and concentric circles or spokes are at 45° angle if k is set to one or minus one; see Figure 7.3 for an octo-radial grid. Note that if the grid is of finite size, the concentric circles can no longer be equidistant, as it is common in the ortho-radial model, instead, the distance between subsequent concentric circles grows exponentially. However, as with the ortho-radial model, we will consider grids of infinite resolution to enable smooth morphs.

For simplicity, let *spoke segment* denote any segment drawn on a ray emanating from the concentric center point c, let *circular segment* denote any segment drawn on a circle with c as center point and let *spiral segment* denote any segment drawn on the logarithmic spiral with $k \in \{-1, 1\}$.

For all proposed morphing strategies the positions of all vertices are considered

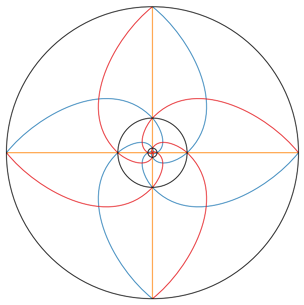


Figure 7.3: An example of the octo-radial finite grid, here with 3 concentric circles shown in orange and 4 spokes in black. Spirals with $k = 1$ are shown in red, while spirals with $k = -1$ are shown in blue. Observe that the distance between subsequent concentric circles grows exponentially.

to be part of the input[1]. This is motivated by the fact that ortho-radial drawings are commonly used for displaying geospatial data. For a morph between Γ and Γ', we linearly interpolate the position of the vertices for every point t in the time interval, i.e. for vertex v the position at point t is $(1-t) \cdot p_\Gamma + t \cdot p_{\Gamma'}$ where $p_\Gamma, p_{\Gamma'}$ denote the position of v in drawing Γ, Γ' respectively. However every edge connecting adjacent vertices v and w has to be represented by a sequence of at most three segments satisfying the constraints for ortho-radial or octo-radial drawings respectively to ensure that at most two bends are used. The individual strategies will construct these sequences. This construction is solely dependent on the relative positions of v, w and the center point c.

7.2 Quality Metrics

In this section, we consider several quality metrics, some of which are well established, others follow naturally from the problem statement. This provides us with a guideline for the design of ortho-radial (and octo-radial) drawing strategies in the following sections. For some metrics, the performance of the individual strategies with regard to the metric is determined solely by design. For other metrics the performance of the strategies are evaluated in the experiment in Chapter 7.4.

Bends. One of the most studied quality metric for planar orthogonal drawings is the number of bends [45, 51, 135]. While an integer-linear program solution exists for minimizing the total number of bends in ortho-radial drawings [96], the algorithm requires the vertices to be freely placeable, which contradicts the geospatial property. Thus we instead consider the number of bends per edge, which has also been extensively studied for planar orthogonal drawings [23, 51, 105]. As we do not restrict the drawings to be crossing-free, one bend per edge suffices for any graph with geospatial data. However using more than one bend can be aesthetically more pleasing, as it allows for more symmetrical representations and beneficial in regards to the other quality metrics.

[1]If geospatial information is not part of the input, any state-of-the-art algorithm (such as spring embedding) can be applied to assign positions to the vertices.

Thus, we will consider one strategy using only one bend per edge and several strategies using at most two bends per edge.

Fréchet distance. For some applications, the straight line representation does inherit some geospatial information, as for instance in metro maps. Thus we investigate the edge-wise similarity between the straight-line and the ortho- or octo-radial representation, as defined by the proposed strategies. A measurement of this similarity is an indicator on how much of this information is preserved during the morph.
One well known measurement for the similarity of curves is the Fréchet distance [3, 52, 69]. Observe that for the edge representations considered, the Fréchet distance between the straight-line-segment and the curve given by the strategies is equivalent to the Hausdorff distance.

Remark 7.1. *For $v, w \in V$ with given positions p_v, p_w. Let ℓ be the straight line segment from p_v to p_w and s any simple curve with endpoints in p_v and p_w. The Fréchet distance between ℓ and s is equal to the Hausdorff distance between ℓ and s.*

Proof. As shown by De Berg et al. [13] the Fréchet Distance between a straight line segment ℓ from p_v to p_w and any curve s is equivalent to

$$\max\{\delta(p_v, s), \delta(p_w, s), \text{hausdorff}(\ell, s)\}$$

where $\delta(p_x, c)$ denotes the minimum distance between point p_x and curve c, while $\text{hausdorff}(c_1, c_2)$ denotes the Hausdorff distance between two curves c_1 and c_2. As both ℓ and s share both endpoints p_v, p_w, it holds that $\delta(p_v, s) = \delta(p_w, s) = 0$. Therefore we have shown the equality

$$\text{frechet}(\ell, s) = \text{hausdorff}(\ell, s)$$

concluding the proof. □

Flips. In a smooth morph the position of vertices and bend points described as a function taking time $t \in [0, 1]$ as input is continuous. However, depending on the drawing strategy, there can exist discontinuities in the placement of bend points. We denote such a discontinuity as a *flip*. Flips impede the tracking of an edge by a user during the morph. Additionally as a flip causes an abrupt visual motion, the user might be distracted and potentially irritated, as such a motion naturally draws the focus of an observer [57]. Thus, the number of flips in a morph should be minimized.

We distinguish between two fundamentally different types of flips, which can occur. *Face flips* occur when the center point passes a straight line segment connecting two adjacent vertices, as this corresponds to a switch of faces in the underlying straight-line drawing. Note that for the proposed strategies a face flip corresponds to a mirroring of the representation of the crossed edge along the axis defined by its straight-line segment. *Style flips* occur each time the drawing style of an edge changes, i.e. any discontinuity which is not connected to a face flip. Figure 7.4 illustrates both types.

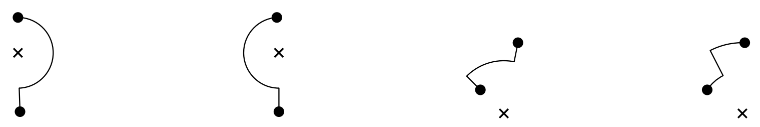


Figure 7.4: An example of a face flip shown in (a) before and (b) after the flip, using the `oneBend` strategy as well as an example of a style flip shown (c) before and (d) after the flip, using the `uniform` strategy.

Port Usage. We also consider the port usage. Generally we consider a stricter limit of available ports as unfavorable, as this increases the number of overlaps and reduces otherwise clear distinction between different edges. However, heavily depending on the application, edge bundling is common for many applications, in which case this quality metric might be of less significance.

Crossings. Finally, we investigate the number of edge crossings. The number of crossings in a graph drawing is well-known to have negative correlation with task performance on the graph [106, 132]. Note, that we do not require the strategies to produce crossing-free drawings. Minimizing the number of crossings during the morph or maintaining planarity should be considered a separate research direction. The vertex placement given might not even allow for a crossing-free drawing and especially the straight-line drawing of the given graph might be non-planar, e.g., in the Sydney graph. Further note that even when the initial drawing is planar, small local or global vertex movements can already prevent a crossing-free drawing, if the number of bend points is limited, see Figure 7.5.
However, depending on the strategy, crossings are more or less likely, thus we evaluate the number of crossings in the experiment. Note that only true crossings are considered by the evaluation, i.e. segment overlaps and segment endpoints placed directly on another segment are not considered crossings, as these deliberately occur in many metro map visualizations.

7.3 Morphing Strategies

Now that we have defined a guiding set of quality metrics, which should be considered in the algorithm design, we will propose several strategies for constructing ortho-radial morphs and one strategy for constructing octo-radial morphs, starting with the ortho-radial strategies.

7.3.1 Ortho-radial

Considering the edge $e = (v, w)$, in polar coordinates let α_v and α_w denote the angles of v and w relative to c and r_v,r_w denote the radii of v and w relative to c, respectively. W.l.o.g. let $\alpha_v < \alpha_w$ and $\alpha_w - \alpha_v \leq \pi$. Further, we assume w.l.o.g. that $r_v < r_w$.

Describing the morphing strategies we differentiate between two types of drawing styles for any given edge with two bends. A *spoke step* denotes an edge representation of e, where the sequence of segments consists of two spoke segments and one circle segment. In this case let r_e denote the radius of the circular

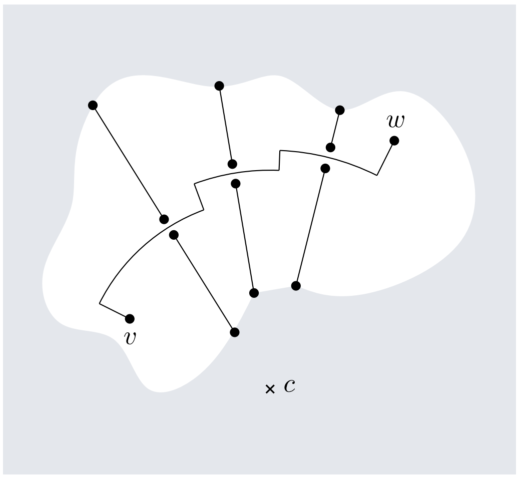


Figure 7.5: The schematic representation of a constellation, where any transposition of the vertices would imply a drawing requiring additional bends per edge to be crossing free. This construction can be repeated.

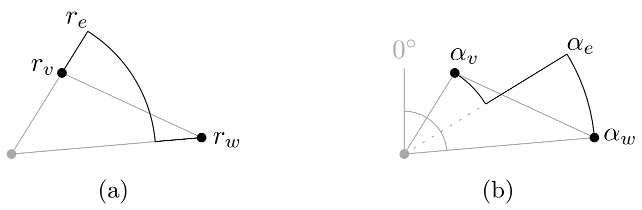


Figure 7.6: A spoke step shown in (a) with radii shown and a circular step shown in (b) with angles shown.

segment. A *circular step* denotes an edge representation of e, where the sequence of segments consists of two circle segments and one spoke segment. In this case let α_e denote the angle of the spoke segment. See Figure 7.6 for instances of a spoke and a circular step.
It suffices to give a definition for r_e and α_e respectively to conclusively describe the sequence of segments, as the positions of v and w are known and the sequence must be consecutive, ending in both v and w.

The two-bend strategies are thus defined by giving a rule to decide which drawing style is used, i.e. whether an edge is represented by circular or spoke steps and a definition for r_e and α_e.

One-bend The most straight-forward drawing style however uses only one bend per edge, thus consisting of only two segments, namely one circular and one spoke segment joined at a bend point, see Figure 7.7. As the positions of both v and w are fixed, there exist exactly two options to place the segments, such that at most 180° are spanned. W.l.o.g suppose that v is closer to the center point, i.e. $r_v \leq r_w$. Now one option is to start with a circular segment incident to v, while the spoke segment is incident to w, the other option reverses the order of the segments. We observe the following.

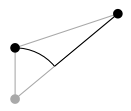
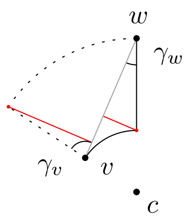
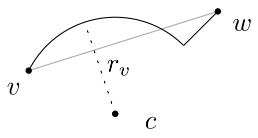


Figure 7.7: Edge representation with `oneBend`.



(a) Schematic showing the distances between bend points and straight line segment depending on whether v or w is incident to the circular segment. The distance is shown in red.



(b) Schematic showing how the distance of the circular segment correlates to the radius, if angle between the straight line segment and the spoke segment at v is greater than 90°.

Figure 7.8

Lemma 7.1. *The Fréchet distance between a one-bend drawing of an edge e and its straight line representation is minimal if the circular segment is incident to the vertex closer to the center point.*

Proof. As we have shown in Remark 7.1, Hausdorff and Fréchet distances are equal for these instances. Thus we will use the definition of the Hausdorff distance for this proof.
Note, that there exist two extreme-point candidates defining the Hausdorff distance: either the bend point, joining both segments, or an interior point of the circular segment. Assuming that the circular segment contributes the defining point, the point must be interior as one of the endpoints coincides with either v or w, thus having a distance of 0 to the straight-line segment while the other coinciding with the bend point, which contradicts the assumption. Observe that the maximum distance of the circular segment to the straight line segment is the point where the straight-line segment is parallel to the tangent of the circular segment, see Figure 7.8b. Thus, the distance directly correlates with the radius of the circular segment. As $r_v < r_w$, this is minimal, exactly if the circular segment is incident to r_v.

Next assume that the bend point defines the Hausdorff distance. We denote the angle γ_v as the angle defined by the spoke segment and the straight line segment incident to v. Analogously we denote γ_w. See Figure 7.8a for an instance displaying these definitions. We observe that, due to $r_v < r_w$, it holds that $\gamma_v > \gamma_w$. Further we observe that if $\gamma_v \leq 90°$ the distance between straight line segment and bend point is

$$\sin(\gamma_i) \cdot (r_w - r_v)$$

where i is v or w, depending on whether the spoke segment is incident to v or w

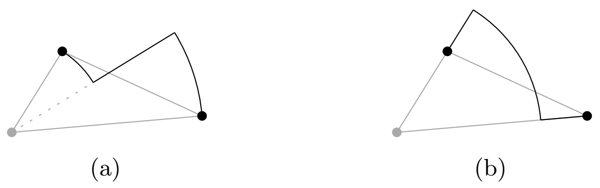


Figure 7.9: Edge representations with `uniform steps`: (a) circular step (b) spoke step.

respectively. Thus, using $\gamma_v > \gamma_w$, it holds that

$$\sin(\gamma_v) \cdot (r_w - r_v) > \sin(\gamma_w) \cdot (r_w - r_v).$$

Therefore the distance is minimized if the spoke segment is incident to w. If $r_v > 90°$ the distance is exactly $(r_w - r_v)$, which is an upper bound for $\sin(\gamma_w) \cdot (r_w - r_v)$, thus the distance is trivially minimal. This concludes the proof. □

We continue with several strategies using two bends per edge.

Uniform steps. While the additional bend per edge increases the total number of bends of the drawing, it also allows for a more adaptable edge representation. One of the most notable psychological aesthetics is symmetry [133], as it is seen as a powerful indicator for beauty. Thus in the *uniform* strategy the edges are represented such that they follow a uniform and symmetric design principle. Additionally, for a more balanced overall aesthetic, the drawing style (between circular and spoke step) is chosen such that the segments are split in a way that a small ratio between largest and shortest segment is maintained. Specifically, for a given edge $e = (v, w)$, the drawing style is a circular step exactly if $r_e \cdot (\alpha_w - \alpha_v) > r_w - r_v$, which is true exactly if the sum of the lengths of circular segments is larger than the sum of the lengths of spoke segments.
To realize the concept of symmetry, let $\alpha_e := \frac{\alpha_v + \alpha_w}{2}$, i.e. α_e is the angle of the bisector of the spokes v and w are placed on, see Figure 7.9a. For spoke steps, let $r_e := \frac{r_v + r_w}{2}$, i.e. r_e is the average radius of the two vertices, see Figure 7.9b.

Fréchet steps. Using the Fréchet steps strategy for a given edge $e = (v, w)$, the drawing style is decided by minimizing the Fréchet distance. Thus, the drawing style is a circular step exactly if the circular step yields a lower Fréchet distance than the spoke step, a spoke step exactly if the spoke step yields a lower Fréchet distance than the circular step and the drawing style of the previous drawing of the morph is maintained, in case of equality.
For circular and spoke steps both r_e and α_e are defined such that the resulting edge representations have minimum Fréchet distance to the straight line segment from v to w. Recall, that for the edge representations considered, the Fréchet distance is equivalent to the Hausdorff distance between the curve and the straight line segment representing an edge, as shown in Remark 7.1. Thus, we use the simpler definition of the Hausdorff distance to make optimality arguments.

Observe that the spoke steps are more flexible than in the `uniform step` strategy, since $r_e < r_v < r_w$ can be true, i.e. the radius of the circular segment is not in between the radii of the endpoints. In particular this is always the case if $r_v = r_w$, as a $r_e > r_v$ would yield a significantly higher Fréchet distance, see Figure 7.10b. Figure 7.10 displays all variants of Fréchet steps and additionally shows the points on the edge representation which determine the Fréchet distance.

We observe that in many cases the Fréchet distance obtained by optimal circular steps and optimal spoke steps is the same:

Lemma 7.2. *If the triangle defined by the vertices v, w and center point c has a $> 90°$ angle at corner v, then the Fréchet distances of both the circular and spoke step are the same.*

Proof. We first consider circular steps. Note that the Fréchet distance is minimal if both bend points are at the same distance to the straight line segment, denoted as s. All other points on the sequence are closer to s. This is the case exactly if the the spoke segment is split halfway, i.e. it intersects with the straight line segment s from v to w exactly at half length of the spoke segment. Thus for the circular step the Fréchet distance is $d = \sin(\alpha_v) \cdot \frac{(r_w - r_v)r_v}{r_w + r_v}$.

Now for the spoke steps, the Fréchet distance is defined by the distance of the two bend points to s. Since the triangle is obtuse at v the points on the line passing through v and w which are closest to the bend points are part of the straight line segment s. Therefore the Fréchet distance d^* of the spoke step is optimal, when both bend points have the same distance to s, thus we get two definitions for the distance $d^* = \delta_v \sin(\alpha_v)$ and $d^* = \delta_w \sin(\alpha_w)$, where δ_v and δ_w denote the differences in radius $|r_e - r_v|$ and $|r_w - r_e|$ respectively. We can combine both definitions to $\delta_v = r_w - r_v - \delta_w$, to replace δ_v with:

$$d^* = (r_w - r_v - \delta_w) \sin(\alpha_v).$$

Replacing δ_w by the second definition given for d^*:

$$d^* = \left(r_w - r_v - \frac{d^*}{\sin(\alpha_w)} \right) \sin(\alpha_v)$$

$$d^* = \frac{(r_w - r_v)\sin(\alpha_v)}{\frac{\sin(\alpha_v)}{\sin(\alpha_w)} + 1}.$$

We aim to show equivalence between d and d^*, thus we replace $\sin(\alpha_w)$ with $\frac{r_v}{r_w} \sin(\alpha_v)$:

$$d^* = \frac{(r_w - r_v)\sin(\alpha_v)}{\frac{r_w}{r_v} + 1} = \frac{(r_w - r_v) r_v \sin(\alpha_v)}{r_w + r_v} = d$$

which concludes the proof. Note that this observation also holds for triangles where v is non-obtuse, however in those cases the definitions given for d and d^* do no longer necessarily correspond to the Fréchet distance between the edge representation and s, for example see Figure 7.10c. □

Due to this property the case of equality often occurs in the implementation, this is resolved by keeping the drawing style of the previous discrete time step in case of equality for any but the first drawing and choosing the drawing style by random in case of equality for the initial drawing of the morph.

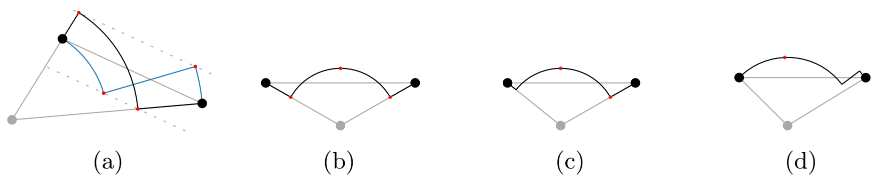


Figure 7.10: Edge representations using the `frechet` strategy: (a) circular step in blue, spoke case in black (b)-(c) spoke cases (d) circular case.

Single style variants. While the number of occurrences depends on the specific strategy, generally some style flips occur when applying the strategies denoted as `uniform` and `frechet`. We already discussed, why style flips are unfavourable, thus we will additionally consider variants which are restricted to one drawing style each, thus preventing all style flips. Namely `uniformcirc` and `frechetcirc` use only circular steps, and `uniformspoke` and `frechetspoke` use only spoke steps as defined in `uniform` and `frechet` respectively. We will discuss in Chapter 7.4 whether these restricted cases show significant drawbacks in terms of crossings or Fréchet distance.

Port by Angle. We further study a strategy inspired by orthogonal algorithms [14], where the slope between incident vertices determines the ports used. We denote this strategy as *port by angle*.

Considering an edge $e = (v, w)$. To decide which one of the four ports is used, the straight line segment between v and w is considered. For one of the vertices, w.l.o.g. v, the plane is divided into four sectors, where the boundaries are lines with $45°$ and $-45°$ angle in relation to spoke grid line v is placed on, see Figure 7.11a. Depending on the sector e is part of the south, west, north or east port is assigned at to e at vertex v. Informally, the edge is assigned the port it is most similar to in a straight line representation. This is done for both endpoints of e individually, resulting in four possible cases.

1. Edge e uses uses the east port at v and the west port at w: In this case we use the circular step construction of the `frechet step` strategy to draw the segment, as it ensures circular segments at both ports, see Figure 7.11b.

2. Edge e uses the east port at v and the south port at w: In this case we construct a sequence of two segments consisting of one circular segment with v as its left endpoint, thus, with radius r_v and spanning from α_v to α_w and one spoke segment, spanning from radius r_v to radius r_w at α_w, thus, having w as an endpoint. Informally, this step resembles the shape of an inverted L, see Figure 7.11c.

3. Edge e uses the north port at v and the south port at w: In this case we use the spoke step construction of the `frechet step` strategy to draw the segments. This ensures that both endpoints are incident to spoke segments, see Figure 7.11d.

4. Edge e uses the south port at both v and w: Informally this case occurs when v and w are on opposite sides of the center point. Necessarily there must be at least a difference in angle of $90°$ between α_v and α_w. To fulfil the

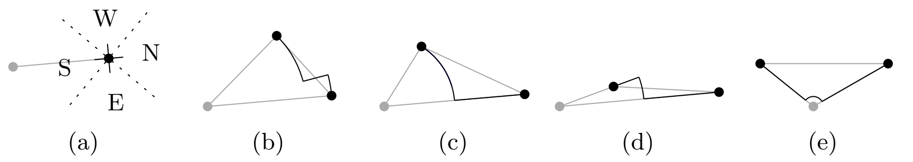


Figure 7.11: Edge representations with `port by angle` strategy: (a) The sectors indicating which port to use. The center point is shown in gray (b) e uses the east port at v and the west port at w. (c) e uses the east port at v and the south port at w. (d) e uses the north port at v and the west port at w. (e) e uses the south port at both v and w.

port constraints, both v and w must be incident to spoke segments, whose endpoints are closer to c. Therefore both spoke segments are assigned endpoints at radius $r_\epsilon << r_v < r_w$. A circle segment at radius r_ϵ from α_v to α_w then joins both segments, see Figure 7.11e. Note that in the experimental implementation we appointed r_ϵ a fixed value.

We observe that no other constellation of designated ports can occur with the given port assignment strategy.

7.3.2 Octo-radial

As with the ortho-radial model, we distinguish two edge drawing styles, however they are slightly adapted to fit the octo-radial model. A *spoke step* consists of two spoke segments and one spiral segment. A *circular step* consists of two circle segments and one spiral segment.
However, since for any edge $e = (v, w)$ the position of the vertices v and w is given, the drawing style representing e is directly implied. This is due to the spiral segment having a fixed slope angle, thus the angle spanned by the spiral segment and the radius spanned by the spiral segment strictly correlate. Therefore, a spiral segment which spans $\alpha_w - \alpha_v$ spans a fixed radius relative to r_v. Let r_e denote this radius. If $r_e \leq r_w - r_v$ then a spoke step can be constructed, however a circular step is impossible. If $r_e \geq r_w - r_v$ vice versa. Both cases are shown in Figure 7.12a and 7.12b. For circular steps, the sequence is constructed such that the circle segments are of equal angle. For spoke steps, the sequence is constructed such that the spoke segments are of equal length.

Figure 7.12: (a) The circular octo-radial case and (b) the spoke octo-radial case.

7.4 Experimental Setup

In this section we experimentally evaluate the proposed algorithm, such that, alongside the theoretical observations, an informative comparison between the

strategies can be made. We implemented all nine strategies using Python and analyzed their performance on the four real world benchmarks as well as the randomly generated data. We remark that the morph animations resulting from all eight strategies were computed in real-time on a desktop PC with 6 cores at 2.8GHz CPU and 16GB RAM. Thus all proposed strategies are suitable for use in practical real-time applications.
We construct several benchmarks on real world data. Additionally we generate random data analyse the resulting morphs under the set of quality metrics introduced in Section 7.2. But first we present a common use-case which will provide a framework for the experiment.

Use-Case

In digital state-of-the-art navigation tools, the shown detail of a map or transportation network is not static but dynamically adapts to the users position. Thus in our hypothetical use-case we consider the movement of the user, i.e. the center point of the visualization, within a network with fixed vertex positions, as bus stations or similar landmarks are static relative to each other. Note that this resembles a transposition of all vertex positions, if the center point is fixed at the origin.
Note that while state-of-the-art applications also support re-scaling and rotation of the displayed network, with an ortho-radial or octo-radial model, both operations are trivial. Regarding the analysis moving the center point has multiple benefits. The lack of (partial) re-scaling allows for comparability between the individual drawings throughout the whole morph, while the transposition of the full network does enforce an update of every edge representation between any two subsequent drawings of the morph.

7.4.1 Real-World Data

For the evaluation, we construct four benchmarks on real-world data, specifically on the Sydney metro map and the Vienna metro map. The networks are chosen to display different characteristics found in real world transportation networks. *Sydney.* The network consists of 34 vertices and 41 edges. The vertex placement is generally more widespread than in the Vienna graph and there exist more large faces in the provided drawing. However there exists one crossing in the straight-line drawing, as two metro lines cross. *Vienna.* The network consists of 24 vertices and 37 edges. The vertex placement results in a generally more condensed drawing than the Sydney graph and includes a dense city center. Note that the network drawn straight-line is planar and that, apart from the outer face, all faces are quadrangles or triangles.

As described in the use-case, the center point is considered to be moving, which is dual to the transposition of the network. Therefore we define two types of trajectories per data set along which the center point moves. A polyline path and a Bézier curve. Specifically, the polyline paths move the center point along the routes of metro lines, thus representing a traversal through the cities using the metro, while the Bézier Curve represent schematic walks from one landmark to another without using the metro. All four benchmarks are shown in Figure 7.13.

Specifically the polyline paths traverse the cities from north to south. On the

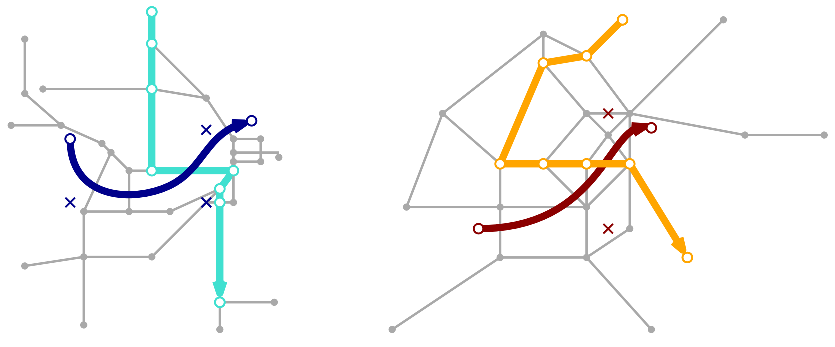


Figure 7.13: The benchmarks (a) Sydney and (b) Vienna with the respective Bézier and Polyline benchmark trajectories.

Sydney graph the polyline path represents a ride from station Berowra to station Sutherland using the the lines T1, T9 and T4. On the Vienna graph the polyline path represents a ride from station Floriansdorf to station Simmering using the lines U6 and U3.

The Bézier curves connect two landmarks of the cities each. On the Sydney graph the Bézier curve represents a walk from the Sydney Zoo to the Opera House. On the Vienna graph the Bézier curve represents a walk from Schloss Schönbrunn to the Prater.

For the evaluation, the individual data points are sampled by moving the center point along the trajectories, constructing the drawings for 201 consecutive positions, using the respective morphing strategy. As every edge representation has to be updated between any pair of frames and the centerpoint visits no position twice there are 8241 different edge representations per benchmark for Sydney benchmarks , while for Vienna there are 7437 different edge representations per benchmark.

7.4.2 Spatial Graph Sampling

Additional to the benchmarks using real-world data we introduce a randomized experimental setup which allows for a larger sample size on unbiased data for a more in-depth analysis. In this section we propose a generator model for randomized spatial networks, random trajectories and explain the details of the respective experimental setup, as we are not aware of any pre-existing framework.

Graph generation The algorithm first generates a set number of $n-3$ vertices, $n \geq 3$, by placing them on the plane, such that their positions are uniformly sampled within the boundaries of an isocles right triangle. To do so, we first uniformly sample positions in a square and mirror their position along the diagonal if they are outside of the triangle described by three vertices of the square. The three vertices of the bounding triangle are added to the vertex set.

Next the vertex set is triangulated without crossings [2]. To provide a general data set we uniformly sampled the inner triangulation of the vertex set. To do so, a random permutation of all $n(n-1)$ edges of the complete graph is generated, then all edges are considered iteratively. If an edge is planar (drawn straight line) considering the current edge set, it is added to the edge set, if it crosses at least one of the edges of the current edge set it is discarded. Note that while this takes $\mathcal{O}(n^3)$ time, the resulting graphs are internally triangulated. Since the vertices of the boundary triangle are included in the vertex set, the outer face is a triangle as well, corresponding to the bounding triangle. Thus there are exactly $3n-6$ edges. Naturally the sampled graph is a maximum planar graph. While random removal of edges would technically represent a more diverse sample of general planar graphs, decreasing the density would reduce the number of edge representations to study and generally lower the number of crossings. Thus we studied maximum planar graphs only. Further note that due to the definition of the boundary triangle all sampled graphs span the same area and thus resolutions and distances between sampled graphs are comparable.

Trajectory generation As with the networks based on real world data we study two types of trajectories based on two different aspects of the use-case.

Edge path trajectories are motivated by the user utilizing the network for movement, i.e. their movement follows the straight line segments of the network. To construct such a trajectory we randomly sample a pair of vertices s and t. Using a breadth-first search a shortest path from s to t on the network is computed. Note that the graph distance is considered, not the Euclidian distance. This path corresponds to a polyline connecting all traversed vertices with straight line segments. This is the sampled edge path trajectory.

In contrast *face path trajectories* are motivated by the user travelling off-grid, i.e. explicitly not using the provided network. To sample such a trajectory we randomly sample a pair of faces f_s and f_t from the set of all faces of the drawing, except for the outer face. We then construct a weak dual graph and find a shortest path from f_s and f_t in the weak dual using breadth-first search. The sampled trajectory is defined by the polyline representing this path, where any bend or end point corresponding to a face f is placed at the geometric center point of f.

For the analysis 100 graphs are sampled, where every graph consists of $n = 100$ vertices and $m = 294$ edges. For every graph one edge path trajectory and one face path trajectory is generated. All morphs are calculated with 101 steps. We then applied all nine strategies on the generated data.

[2]We considered using a Delaunay triangulation to define an edge set for the network, as common use-cases resort to the geospatial properties of the drawing. The Delaunay triangulation corresponds to the fully triangulated network with the minimum aggregated edge length which is a reasonable criteria for the construction of a public transportation network. However, to provide a more general data set we opted to uniformly sample the inner triangulation of the vertex set instead.

Table 7.1: Properties of the morphing strategies

Strategy	Face flips	Style flips	Ports/ vertex
`oneBend`	Yes ✗	No ✓	3
`uniform`	Yes ✗	Yes ✗	4
`uniformcirc`	Yes ✗	No ✓	2
`uniformspoke`	Yes ✗	No ✓	2
`frechet`	No ✓	Yes ✗	4
`frechetcirc`	Yes ✗	No ✓	2
`frechetspoke`	No ✓	No ✓	2
`octoradial`	Yes ✗	No ✓	4
`portByAngle`	Yes ✗	Yes ✗	4

Lists whether a strategy can cause face flips or style flips and the maximum number of ports used per vertex. These properties directly follow from the strategy definitions.

7.5 Results and Discussion

For an impression of the visuals the different strategies produce, see Figure 7.14, where the same frame of an animation is represented using all nine strategies. The full animations and still frames are provided at `https://github.com/TheAnonymousSubmitter/MorphingOrthoradialDrawings`. First we observe that the performance in regards to some quality metrics follows directly from the definitions of the strategies. These results are stated in Table 7.1 and are independent of the experimental findings. Fréchet distances, crossings as well as the exact number of style flips are evaluated using the experimental data, in the following we will discuss the findings in detail.

Style Flips. Figure 7.15a lists the total number of style flips obtained for every combination of real world benchmark and strategy while Figure 7.17a shows the style flips for the randomly sampled data. Observe that six of nine strategies prevent style flips altogether, as shown in Table 7.1, however, some of them do this naively by restricting the styles to one (`frechetcirc`, `frechetspoke`, `uniformcirc` and `uniformspoke`) while others do so while maintaining several styles (`oneBend`, `octoradial`). Only considering the drawings causing style flips, `uniform` performs best. We observe that `frechet` shows noticeably higher numbers of style flips for POLYLINE benchmarks than for BÉZIER benchmarks, however this conspicuity should be regarded with caution, as annotated in the discussion below.

Crossings. The experimental results for the number of crossings are shown in Figure 7.15b and Figure 7.17b respectively. The average number of crossings per drawing is listed per strategy. We observe that both `octoradial` and `oneBend` perform especially well. Surprisingly `frechetcirc` and `frechetspoke` perform better than `frechet`. The `uniform` strategy and its variations underperform, resulting in significantly higher numbers of crossings than `oneBend`.

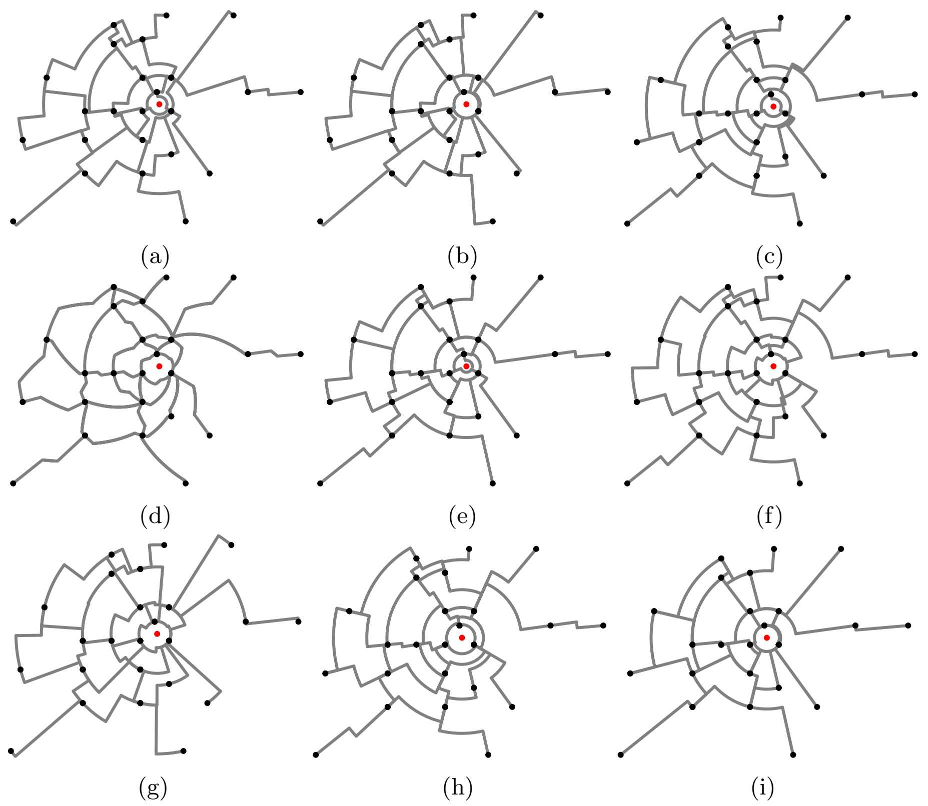


Figure 7.14: Frame 125 of the morph on the Vienna network along a Bézier path using (a) `frechet`, (b) `frechetcirc`, (c) `frechetspoke`, (d) `octoradial`, (e) `portByAngle`, (f) `uniform`, (g) `uniformcirc`, (h) `uniformspoke`, (i) `oneBend`.

Fréchet Distances. The experimental results for the Fréchet distances are displayed in Figure 7.15c and Figure 7.17c. We observe that `frechetcirc`, `frechetspoke` and `frechet` perform very well, with only fractional differences among themselves. The `uniform` strategy and its variations underperform. Even though `octoradial` and `portByAngle` do not optimize curve similarity by construction, they also achieve good results. Notably `oneBend` performs clearly worst regarding the Fréchet distance. Additionally the Fréchet distances were studied in more detail in Figure 7.16b where the variance of a fixed edge is observed during the entirety of a morph. Moreover Figure 7.16a displays the mean Fréchet distances of fixed graphs, aggregated over the morph. As expected the variance correlates directly the Fréchet distances in Figure 7.17c, thus, all of the strategies achieve reasonably stable results.

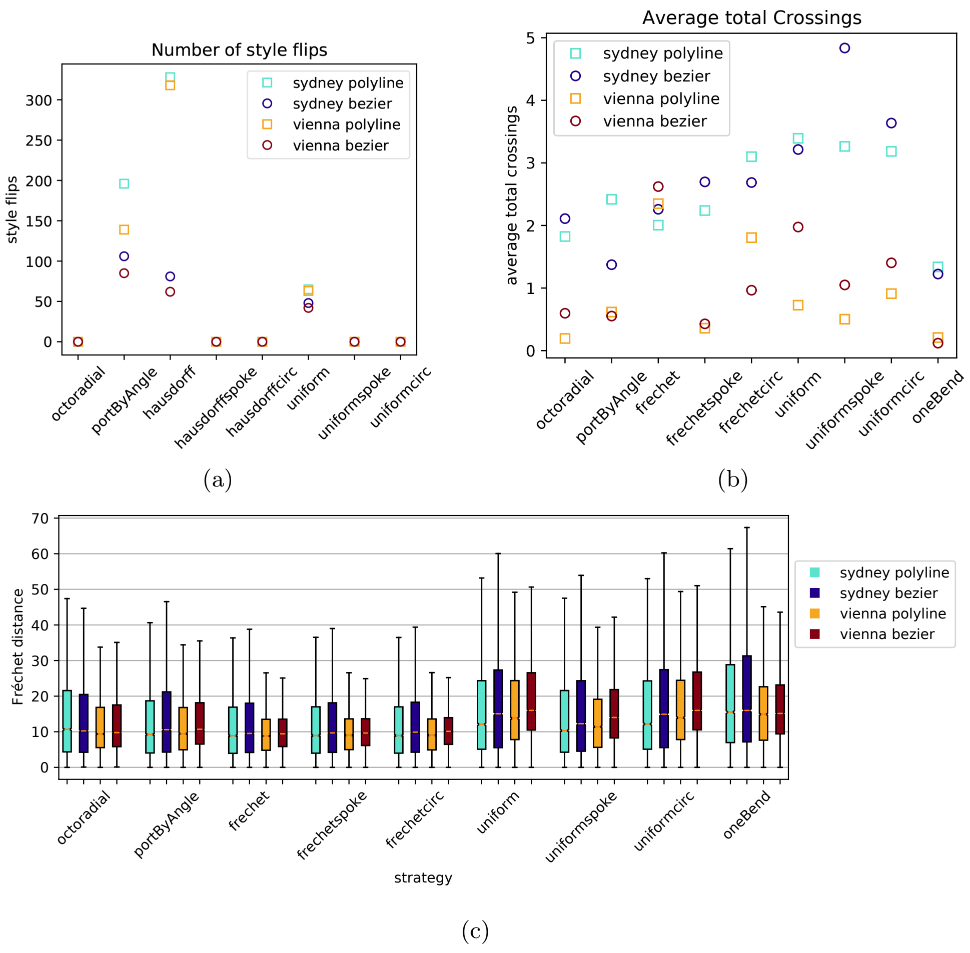
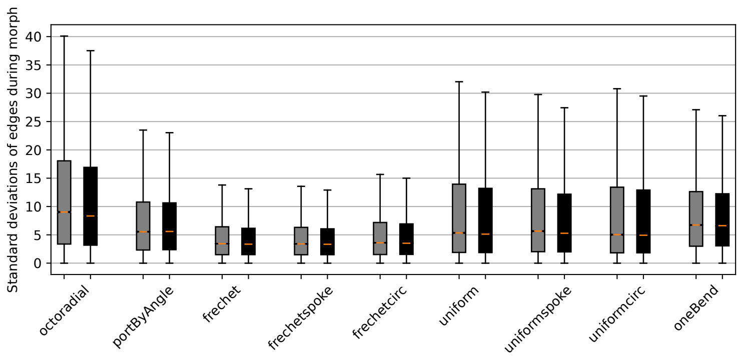


Figure 7.15: (a) The number of style flips, (b) the number of average crossings and (c) the Fréchet distances per strategy and real world benchmark. Note that outliers are not shown for better readability.

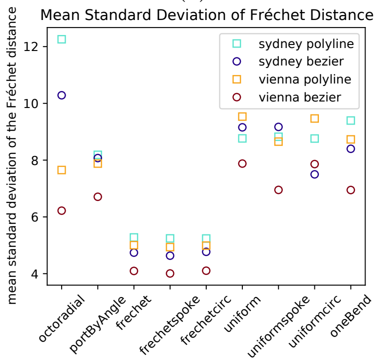
Discussion. As observed above, the number of style flips increases significantly when considering a polyline trajectory with the `frechet` strategy. Note that with a polyline trajectory the center point is on top of a straight line segment representing an edge at any point during the morph. It is likely that this caused a random toggle of the edge style, adding a high number of erroneous style flips. Thus, this entry should be considered with caution.
Considering the performances overall, we propose that `oneBend` and `frechetspoke` are the most versatile and well performing strategies. The simplest approach is the `oneBend` strategy using only one bend per edge and achieving good results regarding the number of crossings while utilizing three ports (the north port is unused). However, it does underperform notably in regard to the Fréchet distance and does not prevent face flips. The `frechetspoke` strategy on the other hand provides completely smooth morphs, while also achieving very good results regarding both the Fréchet distance and the average number of crossings. The primary downside of this strategy is its low port usage. The `octoradial`

strategy receives an honorable mention as it achieves outstanding results in the metrics, however it does not fully utilize the eight available ports of the octo-radial model and due to the spiral segments, users might be less accustomed to the visualization.

As both `oneBend` and `frechetspoke` have unique strengths and drawbacks, they are incomparable. It depends heavily on the application, whether smooth morphs or minimal use of bends and a high port usage are of priority.

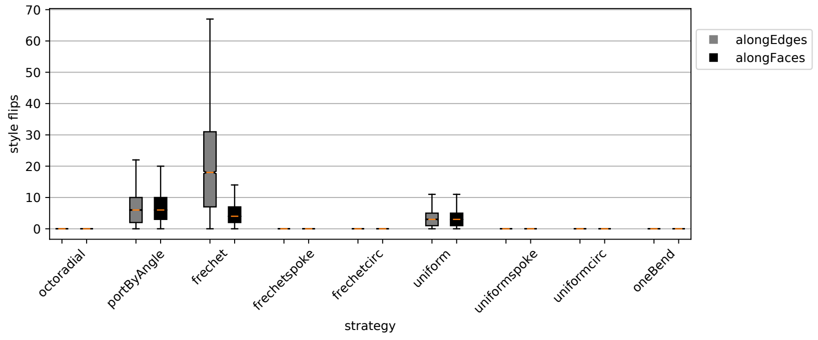


(a)

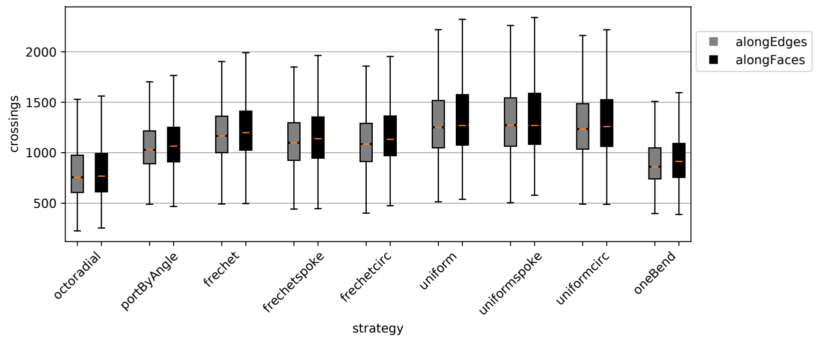


(b)

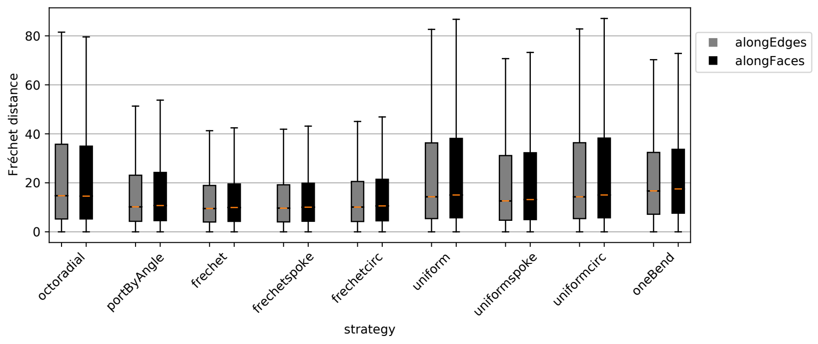
Figure 7.16: (a) The standard deviations of the Fréchet distances of a single edges during the morph. (b) The mean Fréchet distances of all graphs (aggregated over the respective morph).



(a)



(b)



(c)

Figure 7.17: (a) The number of style flips, (b) the number of average crossings and (c) the Fréchet distances per strategy on the randomly sampled data. Note that outliers are not shown for better readability.

Chapter 8

Hybrid Model

The strategies proposed in Chapter 8 are quite general and can be applied in a vast number of use-cases. Here, we study a specific use-case motivated by the everyday problem of using geospatial data for orientation.
State-of-the-art solutions to solve this problem include a Google Maps style representation, where a scalable map is (generally) centered at the user to allow for easy and intuitive orientation, or the more schematic metro-map visualizations widely used to inform about public transportation systems. These models not only aim to provide the user with an efficient way to (manually or digitally) query for common tasks but also provide general information and context to the query result. Thus, ideally the user is not only achieving a short time goal (reaching their destination) but also meanwhile learning about the network (memorizing the local area). In case of public transport and on-foot travelling, static metromaps, as found on bus stops or metro stations and dynamic tools, like Google Maps or Open Street Maps are arguably the most common orientation supporting systems. We denote the former as *metro-map* and the latter as *interactive map* in the following. We omit traditional analogue maps as those are rarely used nowadays.
Comparing those two systems quickly highlights different strengths and weaknesses, as shown in Table 8.1.

While the interactive map does provide for interactivity, queries and off-the-grid orientation, i.e. it helps the user find its way to the public transportation stations, it depicts the map along with the public transportation stations non-schematically. Thus, either the scope of the visualization only shows the very

Table 8.1: Model Comparison

	`interactive map`	`metro-map`	`concentric model`
off-grid orientation	Yes	No	Yes (close to the center)
Interactive	Yes	No	Yes
Schematic	No	Yes	Yes (outside of the center)

Lists the limitations of the two common models `interactive map` and `metro-map` as well as the proposed hybrid `concentric model`.

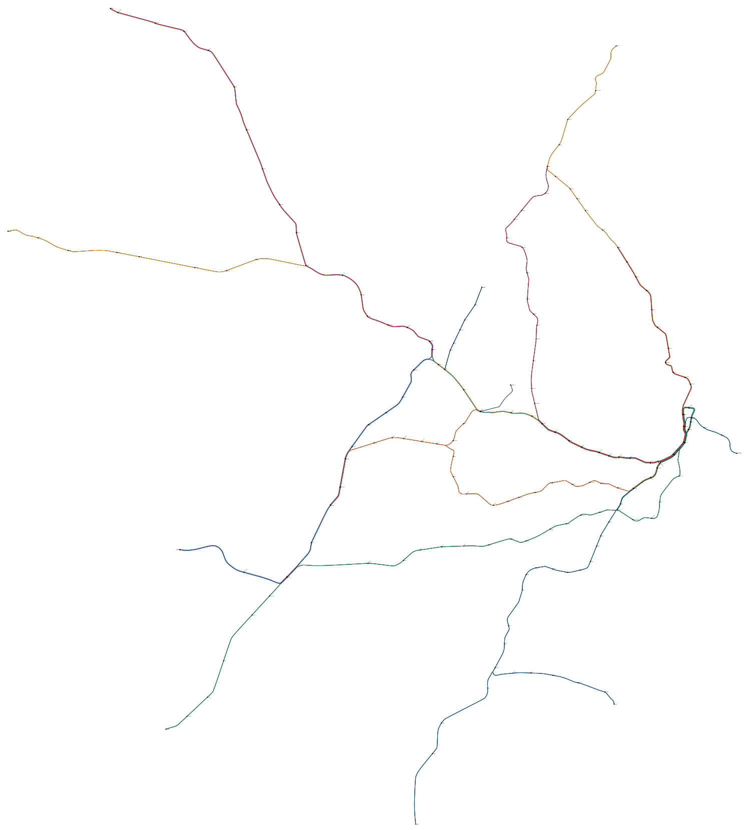


Figure 8.1: The Sydney metro network with correct geospatial proportions. Observe that some parts of the network are not easily readable.

close-by surroundings of the user, omitting much of the public transportation system or the scope is adjusted such that large parts on the transportation system are shown, however this decreases the scale of the displayed map significantly, making it troublesome for the user to gain any local information, especially considering the common screen sizes of hand-held digital devices. Additionally the public transportation system is shown in full detail, showing every twist and turn, which decreases the ability to memorize and learn the system [19]. An additional observation is the unbalanced scale of many real-world public transportation system, as displayed in Figure 8.1 where metro lines leaving the city center can travel great distances, making metro lines in the city center especially hard to read.

In contrast, the metro-map provides almost the opposite usability. It does not show the geospatial distances proportionally, however it does maintain the qualitative geospatial information. This schematic representation of the public transportation network is of good resolution and consists of simpler shapes, thus, it can be viewed completely and can be more easily memorized. However, it is a static visualization, therefore it clearly does not support interactivity or queries. More so, it does not provide any information which could be used for pedestrian orientation. Most importantly it does not provide any support on finding a route to the next (suitable) station.

Based on these observations we propose a hybrid visualization model which aims to combine the benefits of both models while minimizing the shortcomings. We denote this model as the *concentric model*. The model combines a local geospatially exact representation with a globally schematic representation. Specifically it consists of two nested concentric circles. The inner circle provides a sector of the city map, centered at the users position. Public transportation lines are displayed with correct spatial properties within this circle. The outer circle displays only the schematic public transportation network, which seamlessly joins with the lines visible within the inner circle. The network does contain qualitative but not proportionally exact geospatial information, similar to the metro-map visualization.

A visualization of our proposed hybrid model is shown in Figure 8.2.

Thus, the model provides precise information for the local, close-by area, including off-the-grid orientation, while only showing the simplified schematic transportation system for global orientation. As the model is dynamically centered at the users current position, the detailed local information appears and disappears as the user enters and leaves an area. The model also allows for queries, where shortest paths are highlighted as common in the `interactive map` model.

In this chapter, we only propose our hybrid model and discuss technical details and required algorithmic solutions for practical implementation. It is left to future research to conduct user studies on the model to evaluate its practical performance in contrast to the state-of-the-art models. Note that this model directly motivated the research in the previous Chapter 8. The model requires some additional algorithmic steps on top of the previously discussed results, as the morphing strategies are only applied to the (partial) network in the outer circle. We will discuss these technical details along with some optional techniques in the following section.

8.1 Technical Details and Optional Techniques

Split edges. While the ortho-radial model is used to display all network edges which are contained in the outer circle and all edges contained in the inner circle are drawing according to the geospatial information, some public transportation lines might be partially represented in the inner and partially in the outer circle, for instance see lines 2 and 6 in Figure 8.2. For these edges a continuous transition must be guaranteed. Recall that the morphing strategies take the positions of all vertices as an input and preserve these positions, thus the required property can easily be implemented by introducing dummy vertices to every intersection between an edge and the boundary between inner and outer circle. The dummy vertex splits the edge and is positioned exactly at the geospatially exact position of the intersection.

Additionally we propose optional techniques which can be utilized to extend the hybrid model further.

Fisheye. As shown in Figure 8.1 not only the overload of details in the spatially correct visualization causes problems, but also the resolution. However this issue

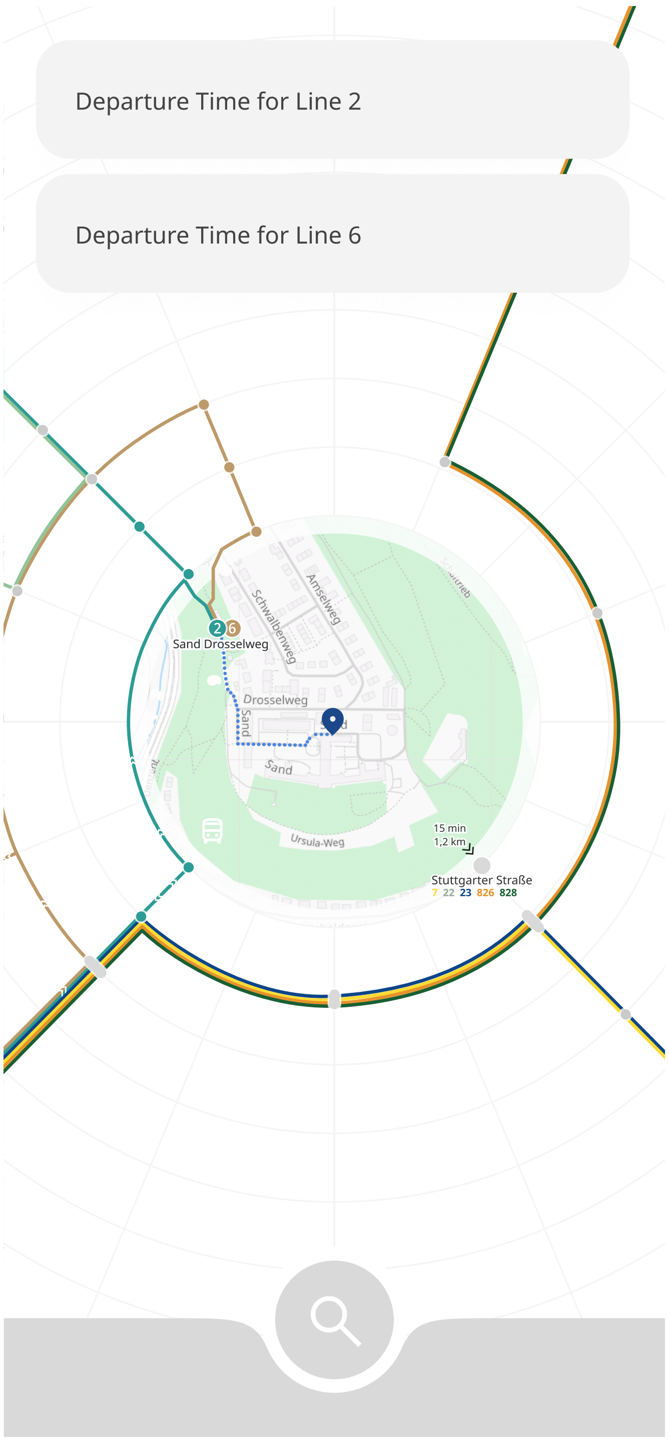


Figure 8.2: The `concentric model` with UI elements.

is not yet addressed by the `concentric model`. As the model aims to provide the most detailed information for the region which is currently closest to the user, we propose a technique which we denote as a *fisheye* processing, which keeps both the resolution as well as the geospatial accuracy of the close-to-center region high, while relaxing both for the network at higher distance. The fisheye preprocessing takes the coordinates of a vertex as an input, converts it to polar coordinates and assigns the vertex a new radius by a *fisheye function* $f : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ taking the original radius as an input. The fisheye function should be monotone and continuous. The continuousness ensures that the smoothness provided by the morphing is maintained. The monotonicity ensures that the correlation between distance to the center point and actual distance is maintained. Additionally the derivation of the fisheye function should be monotonically decreasing to achieve the desired accuracy. Some natural candidates for the fisheye function are logarithmic functions.

Curve Approximation. For the ortho-radial visualization the proposed algorithms take positions of vertices as input, those positions are preserved. However, so far it is undefined what these vertices resemble in the real-world data, as there are multiple natural options. The simplest one takes every station of the transportation network as a vertex, however this can result in a large number of bends as one or two bends in between any two incident vertices are expected, depending on the strategy. Instead it is also possible to define vertices only corresponding to either important stations or stations/points along the route where the direction of the line significantly changes. The stations which do not have a corresponding vertex are then displayed along the visualized path. However the selection of important or significant points along one line would be a manual process.
Instead one can use common curve approximation algorithms from computational geometry, for instance the algorithm by Imai and Iri [78] to select significant points on the lines which are represented by vertices. Even more so, following the motivation of the visualization such an approximation can be adapted dynamically, such that the approximation uses more bends for segments closer to the center point and less bends for further away segments, analogously to the fisheye processing.

Real-time Data. One key feature of the `interactive map` model, making it a state-of-the-art visualization model is the broad support of queries, which the users can easily access. As the proposed hybrid visualization is fundamentally dynamic, it is only natural to support dynamic queries, such as route search and display. Additionally some real-time data should be displayed without the active request by a user, such as the next departure times and lines for close-bye stations. This in combination with the general visualizations provides the user with a large amount of relevant information without the need of any direct user interaction.

Adding an Outer Cycle. Due to the assumptions made by the proposed morphing algorithms they can be straight-forward adapted such that a continuous (however not generally smooth) interlinking with other visualizations is possible, as described in the paragraph "Split Edges". However, this can also be used to combine more than just two visualization models. For instance, if the displayed network is very large and complex, even the schematic visualization might lack in readability. One method to solve this is to add one more concentric circle outside of the ortho-radial circle, which is continuously connected to the ortho-radial model. This outer visualization is even more simplified, such as a labeling of every leaving line, listing all its important stations and omitting any geospatial data.

Chapter 9

Additional Work

In the following we will highlight three practical applications of information visualizations, emphasizing more on the design and the translation of theoretic results to real-world problems and two Graph Drawing challenges, which we solve on a technical, implementation-driven level. Note that all three visualizations are created on real world data, more specifically the second two were entries to the annual Graph Drawing Contest [1] where one won first place in 2022. The first graph presents the findings of a survey of densities of Beyond Planarity classes as a poster, winning the best poster award of the 29th Symposium on Graph Drawing and Network Visualization.
Lastly we will discuss an heuristical approach to multi-criterial optimization, which won in live challenge contests of the annual Graph Drawing Contest 2019, 2020 and 2021. We will shortly describe the techniques used and the approach chosen, while also presenting the achieved results.

9.1 The Universe Beyond Planarity

This work visually presents an overview of the current statues of the research field of beyond planarity as a poster. It displays common beyond planarity classes with known edge density results, inclusion as well as incomparability-relationships between classes as well as recognition algorithms. The poster was elected best poster of the 29th International Symposium on Graph Drawing and Network Visualization. The poster provides a compact and aesthetic summary of the current literature to enable fast lookup of current boundaries, providing an overview and serving educational purposes.

The common classes of beyond-planar graphs are observed in two additional settings, which function as additional requirements to the visualization: The *2-layer* setting, where vertices are required to be placed on either of two parallel lines and edges are drawn in between as well as the *outer* setting, where all vertices have to lie on the outer face. Both settings have received significant study. The classes displayed are the following:

[1]The results of the annual Graph Drawing Contest are archived at https://mozart.diei.unipg.it/gdcontest/

- *k-planar graphs* which are graphs for which a drawing exists such that each edge is crossed at most $k \geq 1$ times.
- *k-quasiplanar graphs* which are graphs for which a drawing exists such that there exist no $k \geq 1$ pairwise crossing edges.
- *fan-planar graphs* which are graphs for which a drawing exists such that no edge is crossed by two edges without a common vertex or by two adjacent edges crossing from different directions.
- *RAC graphs* which are graphs for which a drawing exists such that every edge crossing occurs at 90° angle.
- *IC-planar graphs* are a sub-class of 1-planar graphs. IC-planar graphs are graphs for which a drawing exists such that no two crossing edges share an endpoint, thus they are called *independent*.
- *NIC-planar graphs* are also a sub-class of 1-planar graphs. NIC-planar graphs are graphs for which a drawing exists such that no two pairs of crossing edges share two vertices, thus they are called *nearly independent*.

In the representation, denoted as the *Universe Beyond Planarity*, the three settings are represented by three solar systems. The general setting is yellow, the outer blue and the 2-layer red. Note that the suns themselves represents the ordinary classes of planar, outerplanar and 2-layer planar graphs respectively. The beyond planar graph classes however are represented as planets, where both the orbit and the size of the planet correlate with the known density of the graph class. If several graph classes share the same density aside from additive constants they are shown on the same orbit. Comets represent graph classes with a density nearly identical to the plants they are next too. If the *optimal* version of a class is known, which is the subclass of the class where the number of edges exactly coincides with the density, the optimal subclass is represented as rings circling the corresponding planets. Inclusion relationships between different graph classes are shown with directed edges between planets, while incomparabilities are shown with dashed edges. If an efficient recognition algorithm exists for a graph class, the corresponding representation is shown as being illuminated. This is true for optimal subclasses as well. Additionally there exists one case, where the optimal subclasses of two beyond planar graph classes are identical, which is represented as a connection between the two planet rings.

Figure 9.1 shows the poster.

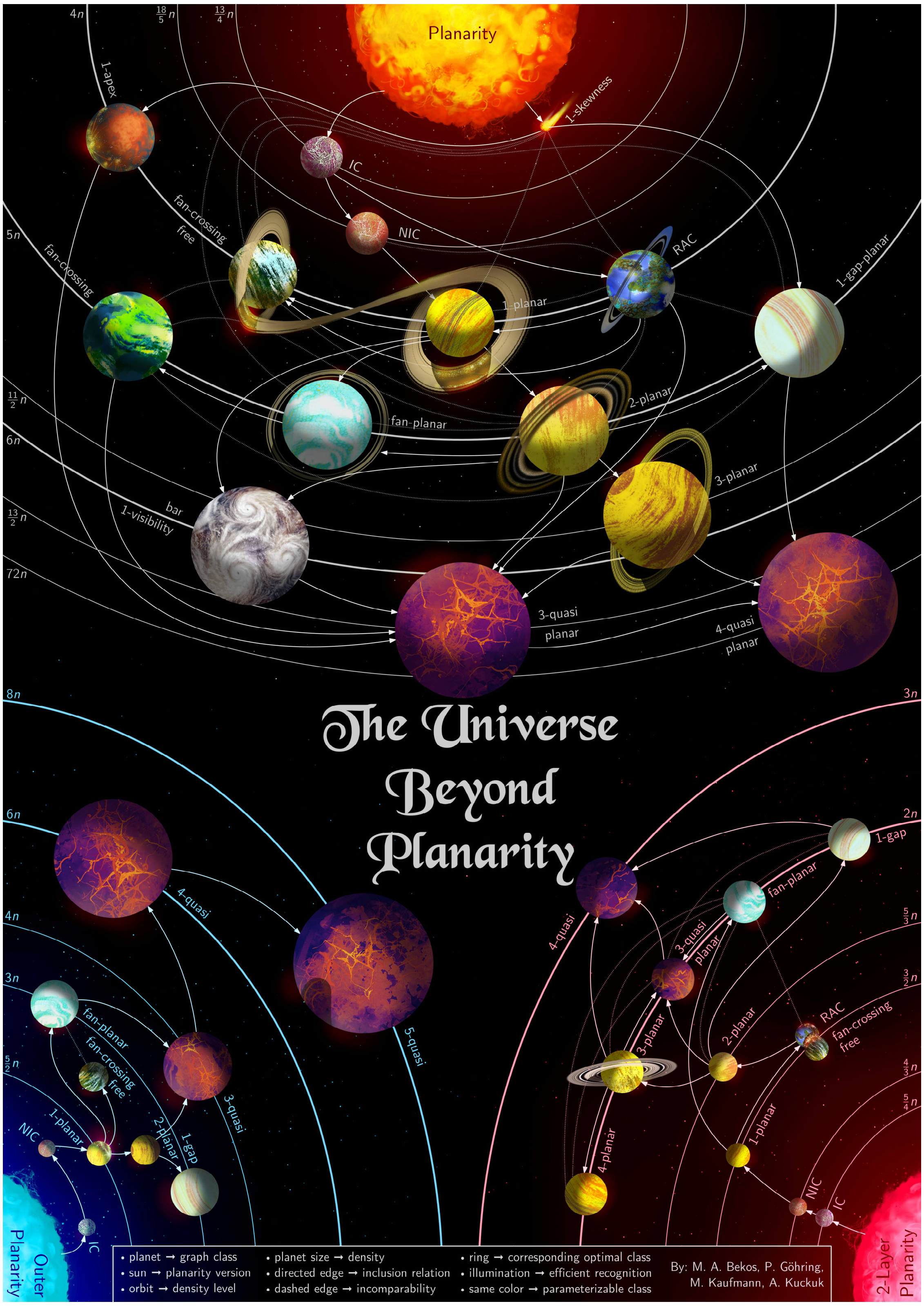


Figure 9.1: A visualization displaying the densities of many known Beyond-Planarity classes for the general, the 2-layer and the outerplanar setting.

9.2 Aesthetic Experience Network

For the 29th Graph Drawing Contest the dataset consisted of the findings of Specker et al. presented in their paper "Associating With Art: A Network Model of Aesthetic Effects" regarding the aesthetic experience of modern artworks [124]. Specker et al. investigated the relation of aesthetic effects, which are measured in sets of two polar characteristics. Subjects considered these characteristics while observing pieces of modern art. The list of effects consisted of: positive — negative, active — passive, still — lively, sad — happy, peaceful — aggressive, hard — soft, cold — warm, light — heavy, rough — smooth, spiritual — bodily, feminine — masculine, cautious — intrusive, like — dislike, interesting — uninteresting. Their research studied the conditional dependence relations between the different effects. Especially it evaluated how (and how much) the relations between effects differ from one painting to the other. Besides the general task to represent the data in a graphical way, the authors were especially interested in "how to visualize this data set for an art historical audience or other audience that does not know about network theory" [33].
Thus besides accurately representing the given data, the visualization aims to encourage the observers to interact in a playful and curious manner with the information without relying on any prior knowledge. To achieve this, we observe the structure of the given data. For any artwork, there exists a complete graph of the characteristics, where the edges are weighted by the correlation between the two characteristics. Note that it suffices to choose one of any pair of characteristics to represent both, as the other characteristic has inverse correlations. We observed that these correlation networks are largely similar between paintings, but show deviations, thus we aim for a dense visualization of every correlation network to enable observers to efficiently perceive and compare the differences between artworks. Also we did not aim to answer the question ourselves, whether there is significant deviation between the networks of different paintings, but provide the tools and prepare the data such that observers can contemplate this question themselves. We were also inspired by the concept of graphical inference as discussed by Wickham et al. in their paper "Graphical Inference for Infovis", where protocols are proposed to compare a visualization against a visualization of random noise to evaluate whether the visualization actually contains information or whether it is just apophenia, where patterns are perceived where there are none and misinformation, which is an important question to be asked in information visualization [134].

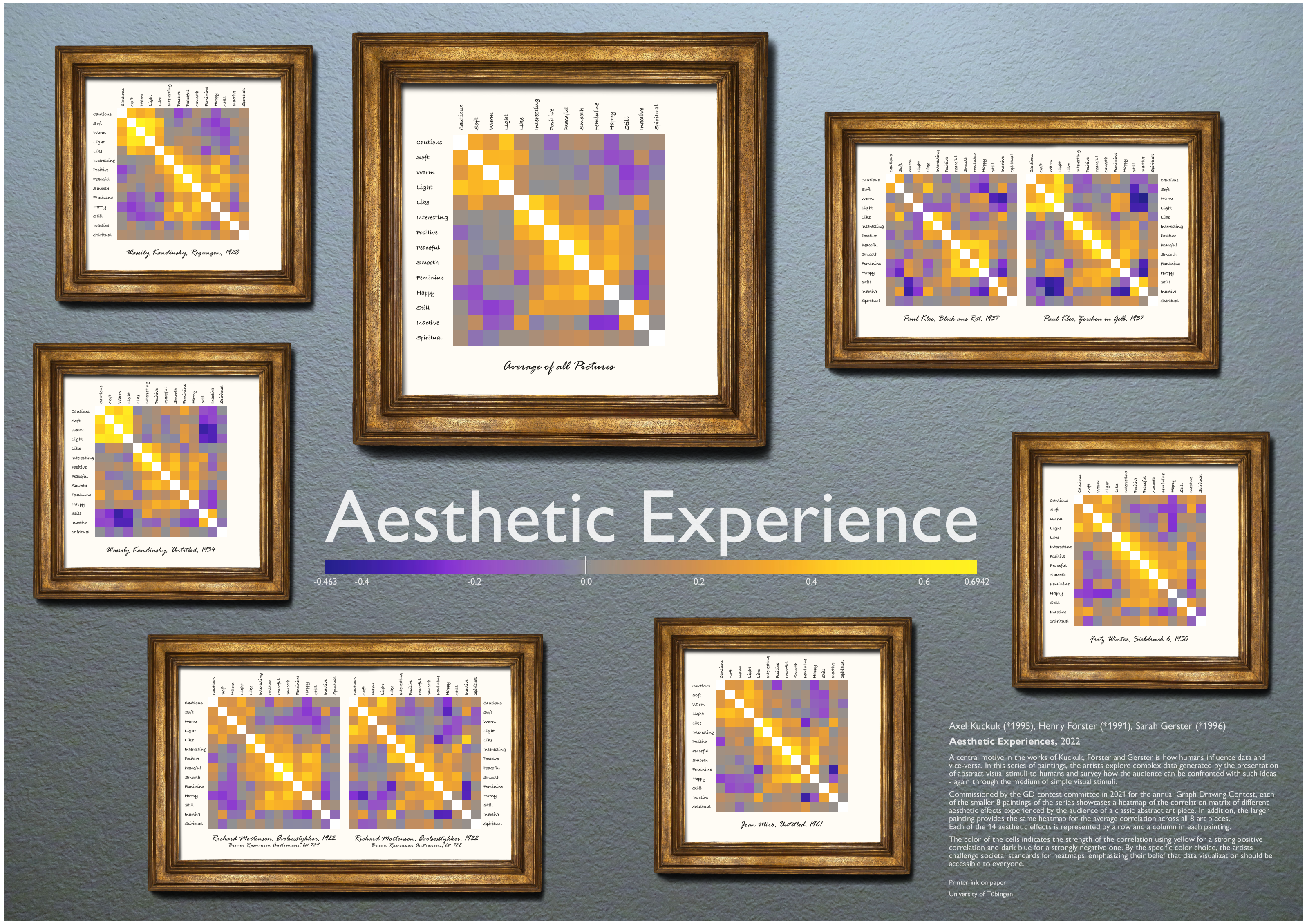
For that purpose we implemented a framework which, given a weighted complete graph and a color-scheme, creates a heat-map displaying the corresponding correlation network. Specifically it creates a 2 dimensional grid heat-map, where every weighted edge, i.e. correlation between two characteristics, corresponds to one cell of the grid which is represented as a color-coded pixel. Thus all data of one painting is represented as one cohesive image.
However, there are multiple pre-processing steps necessary to enhance the readability of the representation. One central step is to reorder the rows and columns of the heat-map to achieve more organic large shapes which highlight clusters. Analyzing the data we find four such clusters, A=(Cautious, Soft, Warm, Light), B=(Like, Interesting, Positive), C=(Peaceful, Smooth, Feminine, Happy), D=(Still, Inactive) while "Spiritual" remains separate. Note that these

five groups of aesthetic effects display high intragroup correlation. Note that some characteristics were inverted (displaying "warm" instead of "cold" for example) such that within clusters the coefficients are (generally) strongly positive. Otherwise high color contrasts within clusters would appear, strongly misleading the observer. Additionally, the average correlation between any pair of characteristics was calculated, including the standard deviation across all networks. In the resulting network high standard deviations of the correlations clearly coaligned with a high average correlation, thus we can assume that the average network represents the typical correlation network appropriately. Therefore the visualization consists of 9 heatmaps, one for each of the 8 artworks and a additional one showing the average of all pictures. This presents the data in a way which is visually evocative of an art gallery, fitting the subject of the visualization.

One additional point to remark is the choice of the color scheme. Clearly a color scheme should ideally be a continual transition between a series of colors, such that the colors at both extreme points are clearly distinct and for all colors in between the appeared color closeness corresponds to the numerical closeness. Optimally the color scheme further serves the aesthetic and thematic demand of the visualization. Furthermore recall that protan and deutan color vision deficiencies are relatively widespread (around 8% of the male population combined [94], thus we desist from using the popular gradient from red over green to blue. Instead the color gradient chosen goes from yellow (representing a strong positive correlation) over magenta to blue (representing a strong negative correlation). We also made sure to choose the gradient such that a correlation of 0 is represented by a neutral grey.

Figure 9.2 shows the final result, It can be observed that often there is a light negative correlation between Cluster A and D, however for some artworks (both paintings of *Paul Klee* and *Wassily Kandinsky, Untitled*) this negative correlation is much stronger. Another example: In *Paul Klee, Blick aus Rot*, Cluster A is by far weaker correlated than in the other works, while the Cluster C rather corresponds to a $K_{2,2}$ (which is displayed by the yellow ring in the corresponding heatmap) instead of a full K_4. Lastly we observe that in comparison with the average heat map, the correlation network of the painting by *Miro* and *Mortensen, lot 729* appears to be the painting invoking the most average aesthetic experiences within the audience.

The poster achieved first place in the 29th Annual Graph Drawing Contest in the Creative Topic category.



Axel Kuckuk (*1995), Henry Förster (*1991), Sarah Gerster (*1996)

Aesthetic Experiences, 2022

A central motive in the works of Kuckuk, Förster and Gerster is how humans influence data and vice-versa. In this series of paintings, the artists explore complex data generated by the presentation of abstract visual stimuli to humans and survey how the audience can be confronted with such ideas - again through the medium of simple visual stimuli.

Commissioned by the GD contest committee in 2021 for the annual Graph Drawing Contest, each of the smaller 8 paintings of the series showcases a heatmap of the correlation matrix of different aesthetic effects experienced by the audience of a classic abstract art piece. In addition, the larger painting provides the same heatmap for the average correlation across all 8 art pieces.
Each of the 14 aesthetic effects is represented by a row and a column in each painting.

The color of the cells indicates the strength of the correlation using yellow for a strong positive correlation and dark blue for a strongly negative one. By the specific color choice, the artists challenge societal standards for heatmaps, emphasizing their belief that data visualization should be accessible to everyone.

Printer ink on paper

University of Tübingen

Figure 9.2: A visualization displaying the experience of an audience observing modern visual art.

9.3 Hrafnkells Saga

For the 28th International Symposium on Graph Drawing and Network Visualization the data set provided for the Graph Drawing Contest was based on the Hrafnkell saga. Hrafnkells saga is an icelandic saga which takes place in the east of Iceland in the 10th century and tells of Hrafnkell who arrives in Iceland as a settler and the events he endures over the years, becoming an atheist and obtaining the rank of a respected chieftain. The provided data consists of one set of actors along with some meta-data like the name, gender and first mention of the character and a set of edges, which represent interactions between two characters, where the source corresponds to the acting person and the sink to the other involved person, as well as a chapter, page and action description. The aim is to present this data visually to the general public [31].

As the data set corresponds to a chronological sequence of events, we decided to use a storyline visualization model [129] to represent the data. In a storyline visualization the actors (characters) of the story are represented by x-monotone curves, which are neighboring at each interaction. The interactions are displayed in time-wise order regarding their x-coordinate. While observing the story in chronological order we make several useful observation.

- Some interactions in the data set serve to describe family relationships between characters instead of actions during the story. We display these interactions separately in the well-known family tree style, where the characters appearing during the course of the story emerge from the family trees.
- Since the set of edges is defined as interactions between two characters, actions which naturally would be considered one action (for instance the scene where Hrafnkell is tortured by Sam and Thorgeirr) consists of several edges ("Person 31 to Person 9, 21-hostility_non-lethal" and "Person 18 to Person 9, 21-hostility_non-lethal"). To enhance readability we merge such actions to interactions involving more than two characters.
- There exist characters which appear in at most two scenes. As those do not really add to the structure conveyed by the storyline, they are not represented by a separate curve but instead are mentioned in the appropriate interactions directly.
- In the data provided, the network was disconnected as there exist characters (for instance Person 27 - milking women) which do not have any incident edges. To resolve this we supplemented the provided data with additional interactions on the basis of the saga.
- Additionally we included more scenes which are essential for the story but did not appear in the provided data. For instance a scene at the start of Chapter 8 in which Thorbjorn considers giving up his case against Hrafnkell, however he is assured in the case by Sam, even though Sam was reluctant at the beginning.
- We rearranged the provided action types, as some of the action types given do not appear in the data ("25 - request information") others appear mostly in pairs ("10 - provide information" and "11 - discover information"),

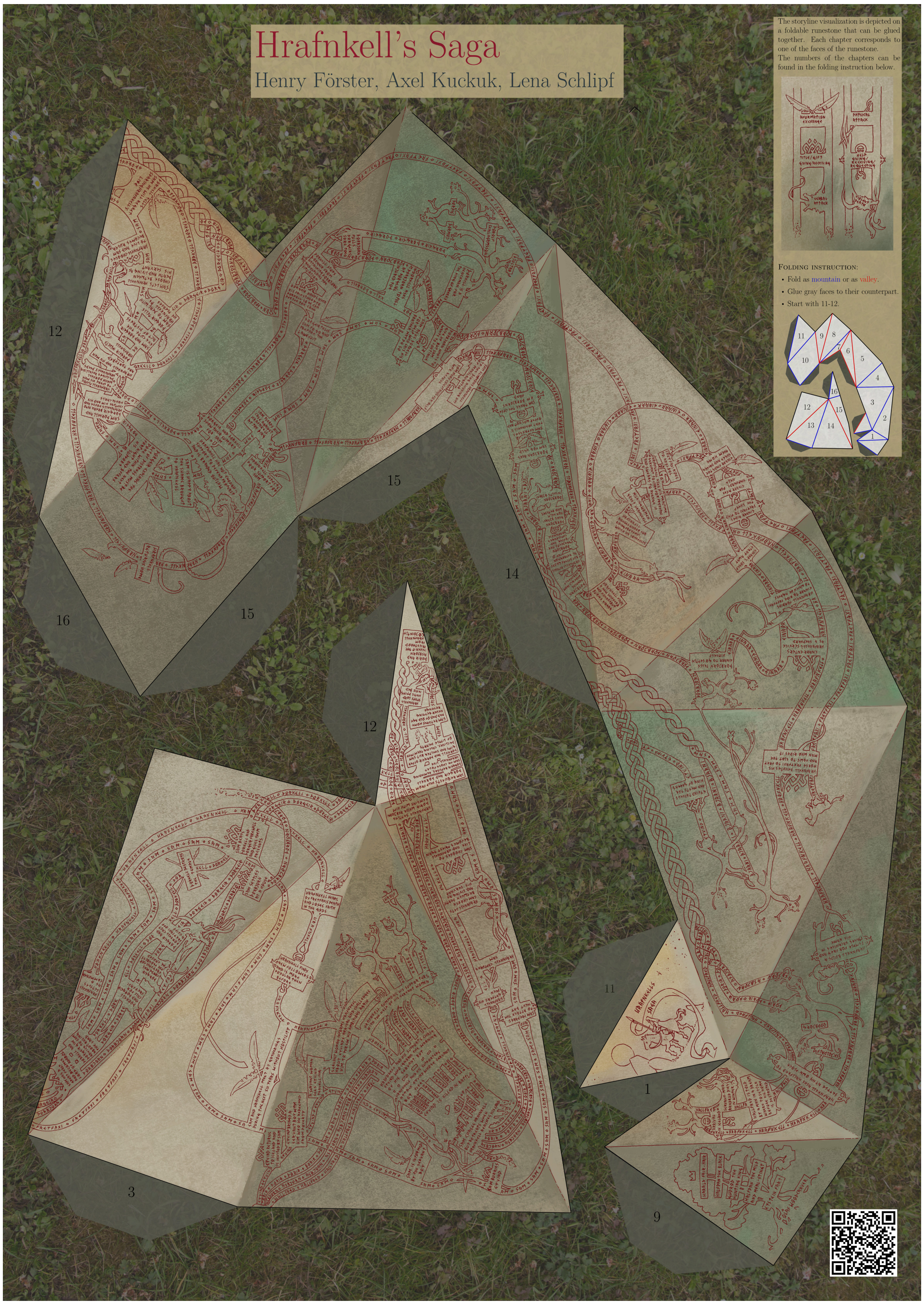
others appear very rarely ("14 - accusation" appears a total of three times). The actions are reordered such that there exist eight clearly distinct categories, specifically: Exchange of Information, Hereditary Information, Peaceful Death, Physical Assault, Providing Gifts, Providing Support, Verbal Assault, Violent Death. These categories are each assigned a graphically mark.

- Lastly, we omitted the page information, as the storyline visualization already depicts the chronological order.

After applying these changes the final data set contained 117 scenes and family relations. Further many curve intersections can be prevented by the orders of appearance and disappearance of the characters.

As the graphical aesthetic plays significant part in sparking the interest of the general public, we choose an unusual design for the storyline representation based on the icelandic theme of the story by presenting it on a runestone. There exist about 3000 runestones in Scandinavia, mostly dated between the 9th and 12th century. The visualization is designed such that the runestone can be folded and glued by interested viewers themselves, if the visualization is printed on a poster. The illustration style imitates that of original runestones, including the use of a strong red color as many original runestones are colored with Falu red. Some additional remarks regarding the visualization: Figure 9.3 shows the final storyline visualization of the Hrafnkell saga.

- To clearly and simply convey the structure of the story each face of the runestone corresponds to one chapter of the story (as well as two additional faces, one for illustrative purposes and one to show the initial family tree of the ruler of Norway at the time of the story).
- These faces are designed such their size is proportional to the number of scenes in the corresponding chapter as well as the importance of the chapter within the context of the story.
- Due to the geometric nature of the runestone, there are faces corresponding to non-consecutive chapters which are still adjacent. This is used to create shortcuts between scenes for characters, which do not appear in intermediate chapters. For instance Chapter 3 is incident to Chapter 13. Multiple characters do appear in Chapter 3 and then again in Chapter 14 without intermediate involvement. This reduces the length of their paths significantly and frees up space within intermediate chapters.
- As most historic runestones are monochrome, using distinct colors to mark the curves representing characters does not fit the intended aesthetic, also such a color-coding relies heavily on the use of a legend, which has to be frequently accessed by the audience. Instead we used a continuous labeling of the curves representing the characters by the character name, which efficiently identifies the character while also resembling the inscription style of some historic runestones.
- To finalize the runestone aesthetic we decided on using runic characters for the lettering. However, the contemporary audience is likely used to the Latin alphabet which makes many runic characters misleading. For instance, the sound of modern day's G can be written as an X. Thus we



The storyline visualization is depicted on a foldable runestone that can be glued together. Each chapter corresponds to one of the faces of the runestone.
The numbers of the chapters can be found in the folding instruction below.

Folding instruction:

- Fold as mountain or as valley.
- Glue gray faces to their counterpart.
- Start with 11-12.

Figure 9.3: A visualization displaying the story of the Hrafnkell saga, a well known icelandic folk saga.

decided on only using specific characters from the Elder Futhark while using a stylized representation of the Latin alphabet otherwise. Note that in some cases this leads to the repurposing of some Elder Futhark characters. For instance to represent the letter P the rune ᚹ (wunjo) is used in the visualization. However ᚹ is originally pronounced like a V or W. Lastly we reintroduced one common runic character, which is ᚦ (thurisaz) representing a th-sound. This character appears in many icelandic names and is still used today in the icelandic language, however with a slightly altered appearance and name (thorn). However we abstained from using the character within non-name words, as to not confuse the modern-day audience.

9.4 Heuristic Crossing Minimization in upward drawings

For the 26th and 27th International Symposium on Graph Drawing and Network Visualization the automatic live challenge was to provide a time-efficient algorithm to reduce the number of crossings within an upward drawing of a given graph. Upward planarity is a useful requirement for visualisation in many use cases, while crossing minimization is one of the most common optimization criteria for Graph Drawing algorithms, as it has been shown that a high crossing number decreases the task performance on a Graph Drawing significantly [106]. Note that the general optimization problem of minimizing edge crossings is NP-hard [74], thus most exact algorithm and heuristics focus on more restrictive cases. One commonly algorithm which is familiar to the provided problem is the Sugyiama framework [126], however this framework requires vertices to be fixed on layers, further it permits bends.

The algorithm we developed and implemented to solve this problem became part of a more general heuristic optimization framework, which can easily adapted to optimize other criteria like crossing resolution, angular resolution and stress and also allows for multicriterial optimization.

The iterative algorithm takes an input of a graph G consisting of n vertices and m edges, then it iteratively creates a series of k valid drawings $\Gamma_1, \dots, \Gamma_k$ within a bounding box of width W and height H. Every graph is evaluated based on an objective function, corresponding to the optimization criterion, in this case the number of crossings. Step $i > 1$ of the algorithm is based on the vertex movement paradigm [59] and works the following: The drawing Γ_{i-1} is already known. Suppose the drawing is not crossing free, then there exist crossing edges. All vertices incident to crossing edges are considered the *vertex-pool* as locally adjusting their position could reduce the total number of crossings.
Then one such vertex v is selected from the vertex-pool at random. The position of this vertex will be locally changed to create drawing Γ_i. All other vertices will be unchanged from Γ_{i-1}. Next we will decide on a set of *candidate positions* which will be considered for the placement of v. This is done by creating a constant number of rays emanating from the current position of v. Then for each ray a random length is chosen (a maximum length is given and might be depending on i, decreasing for later steps of the algorithm), this gives vectors

which determine the candidate positions of v. For every candidate position the validity of the drawing is checked, i.e. whether there is no vertex overlap and no vertices are placed outside of the bounding box, if the drawing is invalid the candidate position is discarded. Additionally, for every candidate position the changes in the objective function are evaluated. The best solution is selected if it improves the optimization criterion or is equal to the current solution. Thus we constructed Γ_i. Note that with a small chance, we also allow for a candidate position to be taken which is strictly worse than the solution of Γ_{i-1}. This allows the algorithm to overcome local optimal solutions.

Note that the performance of the algorithm depends on very efficient iterative steps. Thus some implementation details are crucial to achieve good results. The most important technical improvements we made are the following:

- As the evaluations of the candidate positions are independent, this is an opportunity to use multi-threading. Specifically choosing the number of rays based on the number of processors available greatly increases the performance by parallelization.
- In some cases, if the candidate positions can be evaluated very quickly, it can be more efficient to omit the concept of the vertex-pool altogether and iterate through all vertices repeatedly. This is due to the use of pipelining, which increases the speed significantly, if the vertices which are to be moved are in a fixed order and thus parts of the program can be pre-loaded and even pre-evaluated for a Γ_j, while Γ_i, $i < j$ is not constructed yet. This is not possible, if the vertices are selected by chance.
- For this specific optimization objective crossing calculation makes up a significant part of the running time. Thus it is worthwhile to study the crossing calculation in more detail. Note that the naïve algorithm, calculating all edge crossings takes time $\mathcal{O}(m^2)$. One initial optimization is using a sweep line algorithm to calculate the crossings of the initial drawing in $\mathcal{O}((m+c)\log(m))$ time, where c denote the total number of crossings [11]. Yet, this would only slightly improve the iterative steps of the algorithm.
  However, we can make two important observations. For the evaluation of one candidate position only the edges incident to the moved vertex have to be considered (although they can cross with any other edge) and this change is local and often only affects a fraction of the bounding box, especially for large graphs. Thus we utilize an R-tree data-structure [12], which allows for quick query for overlapping rectangles to find all edges which can possibly be affected by the local change. All other edges do not have to be evaluated. Note that this results in a query-time of $\mathcal{O}(\log(m)+o)$ for edges which might be crossed or no longer crossed, where o denotes the output size while the actual crossing calculation takes $\mathcal{O}(deg(v) \cdot o)$ time, which is significantly faster if o is small.

Remark that while the algorithm works for any computed valid initial drawing Γ_1 the final result and the rate of improvement in regard of the number of iterations depends on the initial drawing. Thus we placed special importance on generating initial drawings which were random and diverse. We further preprocessed those drawings such that the iterative algorithm could work best. To achieve this we used several versions of the Sugiyama framework, while assigning the vertices as uniformly as possible to the layers. We introduced algorithms minimizing the total number of layers the edges span to realize the layer assignment. Note however, that multi-layer edges are not well handled by the Sugiyama framework. Separately we introduced multiple new algorithms based on topological sorting and ear decompositions of the input graphs to generate initial drawings. The preprocessing focused on uniformly distributing the vertices, such that dense areas of the drawing are prevented, as those quickly constrain a central vertex, effectively "locking" it in place, limiting the functionality of the iterative steps.

The algorithm won in the live challenge of the annual Graph Drawing Contests 2019 and 2020.

9.5 Heuristic Edge Length Ration Optimization

For the 28th International Symposium on Graph Drawing and Network Visualization the task of the automatic live challenge was changed. The new objective was to minimize the planar polyline edge-length ratio of a given graph on a fixed grid [32]. The *planar edge-length ratio* of a drawing is the ratio between the Euclidian distance of the most distant adjacent vertices and the Euclidian distance between the closest adjacent vertices of a planar drawing. Note that for the drawings the edges have to be straight-line, thus the ratio corresponds to the ratio of the length of the longest and the shortest edge.

The *planar polyline edge-length ratio* is a variation of the problem, where edges can be drawn as polylines, where the maximum number of bends is given. The ratio is the ratio between the length of the longest polyline and the length of the shortest polyline of the planar drawing. Multiple recent papers study this problem [16, 21, 87].

The heuristical optimization framework described in Section 9.4 can easily be adjusted to optimize the planar polyline edge-length ratio instead. However there are some adjustments to be made:

- The data structure has to be generalized, such that bends are also considered by the algorithm. Note that the movement of a bend only affects the length of one edge.
- As the given graph might not initially contain bends, the algorithm should be able to introduce bends. We decided on only introducing bends, when no further improvement takes place over a set amount of iterations. Note that the algorithm does not remove bends at any points. We observe that the removal of bends on edges is only considerable if the corresponding edge is very short, which is already undesirable given the optimization criterion.
- If an edge has at least one bend, it is no longer possible to change the embedding of the graph by moving one of the incident vertices, as a non-crossing-free intermediate drawing would be necessary. Thus the algorithm included methods of moving full edges instead of single vertices.

However it has to be remarked, that the algorithmic framework is less suited for this optimization problems, as the planarity constrained, combined with bends, restrict the iterative steps considerably. This was evident by the algorithm rarely changing the initial embedding, even though the additional strategy did allow for this. Even more significantly the objective function is at each step of the algorithm defined only by exactly two edges. This either constraints the candidate-pool extremely, which very often leads to local optimal solutions or all vertices have to be considered. However in this case, for all but the candidate-pool vertices, no local objective function exists, as the incident edges of the vertices are independent of the planar polyline edge-length ratio unless they become the longest or shortest edge. While introducing a secondary objective function which locally optimizes the edge-length ratio of all incident edges does improve results in some cases, there is generally much more randomness involved than for other objective functions.

Still, the algorithm won in the live challenge of the annual Graph Drawing Contest 2021.

Chapter 10

Conclusion

In this thesis we made contributions to several theoretical problems, providing algorithmic solutions to some recent challenges in Graph Drawing. To conclude this, we will summarize the results of the thesis and raise related open problems.

Window Width. In Part I of the thesis we introduced the Window Width minimization problem as well as the x-Distance minimization problem. We proved that the Window Width minimization problem is polynomial time solvable if the bottom layer is fixed, providing a sweep-line-algorithm of running time $\mathcal{O}(n_A \log(ww) + m)$. A similar algorithm was used to show that the x-Distance minimization problem is polynomial time solvable if one layer is fixed, yielding a running time of $\mathcal{O}(n_A \log n_A + m)$. Both the Window Width sum minimization and x-Distance sum minimization were shown to be polynomial time solvable as well, reducing the problem to a weighted matching problem with running time $\mathcal{O}(n_A^3 + m)$ each.
In contrast, we proved that the Window Width minimization problem is NP-complete if the top layer is fixed, by reduction from the `Exact 3-SAT` problem. Additionally we provided a polynomial time 2-approximation algorithm for this problem. In Chapter4 we implemented the algorithms and evaluated their improvements on the respective optimization criteria for randomly generated data. We found that there is significant improvement for graphs where there are more vertices of n_A than n_B, especially for sparse graphs. However if nA is smaller than n_B there are rarely significant improvements, implying that other other criteria should be optimized instead.

While the results presented in Part I provided a thorough first attempt to answer the most important questions regarding Window Width and x-Distance in 2 layered drawings, there are still many open questions remaining. The most basic is whether the Window Width can be optimized in polynomial time if no vertices are preassigned to any position. Also a symmetrical definition of Window Width, where not only the Window Widths of vertices of A but also of B should be optimized simultaneously would be a natural variant of the studied problem. Further relaxing the restrictions of the preassigned vertices, where only an order of vertices for each layer, but not a fixed placement is given may be worthwhile. Especially for the NP-hard problem with fixed top layer, one might consider the bottom layer to be ordered but not placed and ask whether this problem is still NP-hard.

Simultaneous Embedding of Multiple Trees. In Part II of the thesis the Crossing Minimization of Upward Trees (CMUT) problem was introduced, where the leaves of k layered rooted trees have to obtain a fixed order. While it is a known result that the problem is NP-complete for an arbitrary number of trees even on just two layers, we proved that the problem is fixed parameter linear for the number of trees k in Chapter 6. To do so, we reduced the problem to a shortest path problem on a k-dimensional weighted cube graph. By extending the algorithmic concept we further proved that the CMUT problem is in XP taking the number of trees k as parameter when considered on three layers. The proposed algorithm reduced the problem to a shortest path problem on a k-dimensional weighted grid graph. Several generalizations of the problem were provided, taking partial orders for the second and third layer as part of the input and extending the result to k planar upward graphs if the third layer is sparse. In Chapter 6 an arbitrary number of layers was considered, however, limiting the number of trees strictly to two. Doing so, we were able to prove that the number of crossings can be minimized in polynomial time $\mathcal{O}(n_1^2 \cdot n_2)$, where n_1 and n_2 denote the number of vertices of the two trees respectively. We generalized this result, taking partial orders as input for all layers, instead of a total order of all leaves. Further we extended the result to one tree and one upward planar graph.

The results approached the CMUT problem from two directions, by restricting either the height or the number of trees. However there is a gap between the two main results which remains the central open question, specifically: is the problem for 3 trees on 4 layers NP-hard? If it is polynomial time solvable, what are the parameters such that the problem transitions into an NP-hard problem? Another open question regards the prerequisites of the drawing. If it is possible to show that the crossing minimal drawing is always planar regarding the individual trees (assuming that the leaf order allows for a planar drawing) the results would be more general. While we conjecture this to be true, so far we were unable to prove it.
Relaxing the prerequisites for a forest to allow for multiple planar or general graphs is also a direction worth studying, as results would potentially influence the state-of-the-art for Sugyiama style drawings.

Ortho-Radial Morphing. In Part III of the thesis we made several advances to make the ortho-radial model suitable for dynamic use. We proposed eight strategies taking geospatial networks as input and constructing ortho-radial representations which allow for continuous updates. In Chapter 8 we introduced those strategies in detail, discussing their theoretical advantages and disadvantages in regards of several well-established quality metrics. Additionally, we introduced the octo-radial model, allowing for 45° spiral segments, extending the the circular and straight line segments of the ortho-radial model which makes the model more flexible and closer to the common octilinear metro map representation. An octo-radial morphing strategy was provided as well. In Chapter 7.4 we put these strategies to the test, applying them to both benchmarks utilizing real-world data and randomly generated data. The resulting number of flips, crossings and the Fréchet distances in respect to the straight line segments were presented and discussed. Considering their overall performances, we proposed the `oneBend` and the `frechetspoke` strategies as the most versatile and capable solutions. In Chapter 8 a hybrid visualization model was introduced combining schematic and exact geospatial representations utilizing the results of the previous chapters.

As the results can be considered the first advance towards morphing of ortho-radial drawings, there remains much to be studied. One challenge is to find algorithms which preserve simplicity and planarity during the morph, assuming that start and end embeddings are equivalent and planar. Considering both the morphing models and the proposed hybrid visualization model a user study would allow a direct comparison with the state-of-the-art models and an investigating into the readability of the resulting drawings and morphs. Regarding the new octo-radial model, strategies utilizing all eight ports may be beneficial for some use-cases and thus form an intriguing open problem.

Bibliography

[1] Albert, R., Barabási, A.L.: Statistical mechanics of complex networks. Reviews of modern physics **74**(1), 47 (2002)

[2] Altenhoff, A.M., Gil, M., Gonnet, G.H., Dessimoz, C.: Inferring hierarchical orthologous groups from orthologous gene pairs. PloS one **8**(1), e53786 (2013)

[3] Aronov, B., Har-Peled, S., Knauer, C., Wang, Y., Wenk, C.: Fréchet distance for curves, revisited. In: Algorithms–ESA 2006: 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006. Proceedings 14. pp. 52–63. Springer (2006)

[4] de Assis Mota, A., Mota, L.T.M.: Drawing meshed one-line diagrams of electric power systems using a modified controlled spring embedder algorithm enhanced with geospatial data. Journal of Computer Science **7**(2), 234–241 (2011)

[5] Barth, L.: Drawing metro maps on concentric circles. Unpublished doctoral dissertation, Master's thesis]. Fakultät für Informatik, Karlsruher Institut für (2016)

[6] Barth, L., Niedermann, B., Rutter, I., Wolf, M.: Towards a topology-shape-metrics framework for ortho-radial drawings. arXiv preprint arXiv:1703.06040 (2017)

[7] Bast, H., Brosi, P., Storandt, S.: Metro maps on flexible base grids. In: Hoel, E., Oliver, D., Wong, R.C., Eldawy, A. (eds.) Proceedings of the 17th International Symposium on Spatial and Temporal Databases, SSTD 2021, Virtual Event, USA, August 23-25, 2021. pp. 12–22. ACM (2021). https://doi.org/10.1145/3469830.3470899, `https://doi.org/10.1145/3469830.3470899`

[8] Bastert, O., Matuszewski, C.: Layered drawings of digraphs. In: Drawing Graphs: Methods and Models, pp. 87–120. Springer (2001)

[9] Bekos, M.A., Förster, H., Kaufmann, M., Kobourov, S., Kryven, M., Kuckuk, A., Schlipf, L.: On the 2-layer window width minimization problem. In: International Conference on Current Trends in Theory and Practice of Computer Science. pp. 209–221. Springer (2023)

[10] Bennett, C., Ryall, J., Spalteholz, L., Gooch, A.: The aesthetics of graph visualization. In: CAe. pp. 57–64 (2007)

[11] Bentley, Ottmann: Algorithms for reporting and counting geometric intersections. IEEE Transactions on computers **100**(9), 643–647 (1979)

[12] Bentley, J.L., et al.: Decomposable searching problems. Inf. Process. Lett. **8**(5), 244–251 (1979)

[13] de Berg, M., Mehrabi, A.D., Ophelders, T.: Data structures for Fréchet queries in trajectory data. In: Gudmundsson, J., Smid, M.H.M. (eds.) Proceedings of

the 29th Canadian Conference on Computational Geometry, CCCG 2017, July 26-28, 2017, Carleton University, Ottawa, Ontario, Canada. pp. 214–219 (2017)

[14] Biedl, T., Kant, G.: A better heuristic for orthogonal graph drawings. Computational Geometry **9**(3), 159–180 (1998)

[15] Binucci, C., Chimani, M., Didimo, W., Gronemann, M., Klein, K., Kratochvíl, J., Montecchiani, F., Tollis, I.G., et al.: Algorithms and characterizations for 2-layer fan-planarity: From caterpillar to stegosaurus. Journal of Graph Algorithms and Applications **21**(1), 81–102 (2017)

[16] Blažej, V., Fiala, J., Liotta, G.: On the edge-length ratio of 2-trees. In: International Symposium on Graph Drawing and Network Visualization. pp. 85–98. Springer (2020)

[17] Bodlaender, H.L., Fomin, F.V., Koster, A.M., Kratsch, D., Thilikos, D.M.: A note on exact algorithms for vertex ordering problems on graphs. Theory of Computing Systems **50**(3), 420–432 (2012)

[18] Booth, T.L.: Sequential machines and automata theory. (No Title) (1967)

[19] Borkin, M.A., Vo, A.A., Bylinskii, Z., Isola, P., Sunkavalli, S., Oliva, A., Pfister, H.: What makes a visualization memorable? IEEE Transactions on Visualization and Computer Graphics **19**(12), 2306–2315 (2013). https://doi.org/10.1109/TVCG.2013.234

[20] Börner, K., Teichmann, S.A., Quardokus, E.M., Gee, J.C., Browne, K., Osumi-Sutherland, D., Herr, B.W., Bueckle, A., Paul, H., Haniffa, M., et al.: Anatomical structures, cell types and biomarkers of the human reference atlas. Nature cell biology **23**(11), 1117–1128 (2021)

[21] Borrazzo, M., Frati, F.: On the planar edge-length ratio of planar graphs. arXiv preprint arXiv:1908.03586 (2019)

[22] Brandes, U., Köpf, B.: Fast and simple horizontal coordinate assignment. In: International Symposium on Graph Drawing. pp. 31–44. Springer (2001)

[23] Bruckdorfer, T., Kaufmann, M., Montecchiani, F., et al.: 1-bend orthogonal partial edge drawing. J. Graph Algorithms Appl. **18**(1), 111–131 (2014)

[24] Buchin, K., Buchin, M., Byrka, J., Nöllenburg, M., Okamoto, Y., Silveira, R.I., Wolff, A.: Drawing (complete) binary tanglegrams: hardness, approximation, fixed-parameter tractability. Algorithmica **62**, 309–332 (2012)

[25] Butler, M.A., King, A.A.: Phylogenetic comparative analysis: a modeling approach for adaptive evolution. The american naturalist **164**(6), 683–695 (2004)

[26] Chan, H.Y., Xu, Y., Chen, A., Liu, X., Cheung, K.K.C.: Drawing metro maps in concentric circles: A designer-in-the-loop approach with visual examples. Transactions in GIS (2022)

[27] Chang, Y.: Ortho-radial drawing in near-linear time. CoRR **abs/2305.00425** (2023). https://doi.org/10.48550/arXiv.2305.00425, `https://doi.org/10.48550/arXiv.2305.00425`

[28] Chiswell, I., Hodges, W.: Mathematical logic. OUP Oxford (2007)

[29] Chung, F.R., et al.: Probabilistic Combinatorics and Its Applications, vol. 44. American Mathematical Soc. (1991)

[30] Ciudad, S.B.A.: Buenos aires: Network map (2023), `https://buenosaires.gob.ar/subte/mapa-del-subte-y-combinaciones`. [Accessed on: 2023-12-31]

[31] Committe, G.D.C.: 28th graph drawing contest - creative topics (2021), `https://mozart.diei.unipg.it/gdcontest/contest2021/index.php?id=creative-topics`. [Accessed on: 2023-12-12]

[32] Committe, G.D.C.: 28th graph drawing contest - live challenge (2021), `https://mozart.diei.unipg.it/gdcontest/contest2021/index.php?id=live-challenge`. [Accessed on: 2023-12-12]

[33] Committe, G.D.C.: 29th graph drawing contest - creative topics (2022), `https://mozart.diei.unipg.it/gdcontest/contest2022/topics.html`. [Accessed on: 2023-12-12]

[34] Consortium, H.: CCF ASCT+B Reporter (2010), `https://hubmapconsortium.github.io/ccf-asct-reporter/`. [Accessed on: 2023-12-19]

[35] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms. MIT press (2022)

[36] Cornelsen, S., Karrenbauer, A.: Accelerated bend minimization. In: International Symposium on Graph Drawing. pp. 111–122. Springer (2011)

[37] Czabarka, É., Székely, L.A., Wagner, S.G.: A tanglegram Kuratowski theorem. J. Graph Theory **90**(2), 111–122 (2019). https://doi.org/10.1002/jgt.22370

[38] De Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. Combinatorica **10**, 41–51 (1990)

[39] Design, B.S.V., DIEINFORMATIONSDESIGNER: Bahnen in köln (2023), `https://www.kvb.koeln/fahrtinfo/liniennetzplaene.html`. [Accessed on: 2023-12-31]

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

[41] Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph drawing: algorithms for the visualization of graphs. Prentice Hall PTR (1998)

[42] Di Battista, G., Tamassia, R., Tollis, I.G.: Area requirement and symmetry display in drawing graphs. In: Proceedings of the fifth annual symposium on Computational geometry. pp. 51–60 (1989)

[43] Di Giacomo, E., Didimo, W., Eades, P., Liotta, G.: 2-layer right angle crossing drawings. Algorithmica **68**(4), 954–997 (2014)

[44] Didimo, W., Eades, P., Liotta, G.: Drawing graphs with right angle crossings. In: Algorithms and Data Structures: 11th International Symposium, WADS 2009, Banff, Canada, August 21-23, 2009. Proceedings 11. pp. 206–217. Springer (2009)

[45] Didimo, W., Kaufmann, M., Liotta, G., Ortali, G.: Computing bend-minimum orthogonal drawings of plane series–parallel graphs in linear time. Algorithmica pp. 1–62 (2023)

[46] Didimo, W., Liotta, G., Montecchiani, F.: A survey on graph drawing beyond planarity. ACM Computing Surveys (CSUR) **52**(1), 1–37 (2019)

[47] Diestel, R.: Graph Theory, 5th Edition, vol. 173. Springer (2017)

[48] Dujmović, V., Fellows, M., Hallett, M., Kitching, M., Liotta, G., McCartin, C., Nishimura, N., Ragde, P., Rosamond, F., Suderman, M., et al.: On the parameterized complexity of layered graph drawing. In: Algorithms—ESA 2001: 9th Annual European Symposium Århus, Denmark, August 28–31, 2001 Proceedings 9. pp. 488–499. Springer (2001)

[49] Dujmovic, V., Fellows, M., Hallett, M., Kitching, M., Liotta, G., McCartin, C., Nishimura, N., Ragde, P., Rosamond, F., Suderman, M., et al.: A fixed-parameter approach to 2-layer planarization. Algorithmica **45**, 159–182 (2006)

[50] Eades, P., Wormald, N.C.: Edge crossings in drawings of bipartite graphs. Algorithmica **11**, 379–403 (1994)

[51] Eiglsperger, M., Fekete, S.P., Klau, G.W.: Orthogonal graph drawing. In: Drawing Graphs: Methods and Models, pp. 121–171. Springer (2001)

[52] Eiter, T., Mannila, H.: Computing discrete fréchet distance (1994)

[53] Erdős, P., Rényi, A., et al.: On the evolution of random graphs. Publ. math. inst. hung. acad. sci **5**(1), 17–60 (1960)

[54] Fernau, H., Kaufmann, M., Poths, M.: Comparing trees via crossing minimization. J. Comput. Syst. Sci. **76**(7), 593–608 (2010). https://doi.org/10.1016/j.jcss.2009.10.014

[55] Förster, H., Kaufmann, M.: On compact rac drawings. In: 28th Annual European Symposium on Algorithms (ESA 2020). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2020)

[56] Foster, E., Towle Jr, B.: Software engineering: a methodical approach. Auerbach Publications (2021)

[57] Franconeri, S.L., Simons, D.J.: Moving and looming stimuli capture attention. Perception & psychophysics **65**(7), 999–1010 (2003)

[58] Frank, F., Kaufmann, M., Kobourov, S., Mchedlidze, T., Pupyrev, S., Ueckerdt, T., Wolff, A.: Using the metro-map metaphor for drawing hypergraphs. In: International Conference on Current Trends in Theory and Practice of Informatics. pp. 361–372. Springer (2021)

[59] Frick, A., Ludwig, A., Mehldau, H.: A fast adaptive layout algorithm for undirected graphs (extended abstract and system demonstration). In: Graph Drawing: DIMACS International Workshop, GD'94 Princeton, New Jersey, USA, October 10–12, 1994 Proceedings 2. pp. 388–403. Springer (1995)

[60] Gansner, E.R., Hu, Y., Kobourov, S.G.: Gmap: Drawing graphs as maps. In: Graph Drawing: 17th International Symposium, GD 2009, Chicago, IL, USA, September 22-25, 2009. Revised Papers 17. pp. 405–407. Springer (2010)

[61] Garey, M.R., Johnson, D.S.: Crossing number is NP-complete. SIAM Journal on Algebraic Discrete Methods **4**(3), 312–316 (1983)

[62] Geiger, R.L., Allen, P.E., Strader, N.R.: Vlsi design techniques for analog and digital circuits (1990)

[63] Gilbert, E.N.: Random graphs. The Annals of Mathematical Statistics **30**(4), 1141–1144 (1959)

[64] Gilbreth, F.B., Gilbreth, L.M.: Process charts: First steps in finding the one best way to do work. Journal of Fluids Engineering **43**, 1029–1043 (1921)

[65] GmbH, V.N.A.D.: Liniennetzplan stadtnetz tübingen (2023), `https://www.swtue.de/oepnv/fahrplan-und-liniennetz/liniennetz.html`. [Accessed on: 2023-12-31]

[66] Goldberg, A.V., Harrelson, C.: Computing the shortest path: A search meets graph theory. In: SODA. vol. 5, pp. 156–165 (2005)

[67] Gutwenger, C., Mutzel, P.: An experimental study of crossing minimization heuristics. In: Graph Drawing: 11th International Symposium, GD 2003 Perugia, Italy, September 21-24, 2003 Revised Papers 11. pp. 13–24. Springer (2004)

[68] Haeckel, E.H.P.A.: Systematische phylogie, vol. 3. G. Reimer (1895)

[69] Har-Peled, S., Raichel, B.: The fréchet distance revisited and extended. ACM Transactions on Algorithms (TALG) **10**(1), 1–22 (2014)

[70] Hasheminezhad, M., Hashemi, S.M., Tahmasbi, M.: Ortho-radial drawings of graphs. Australas. J Comb. **44**, 171–182 (2009)

[71] Hasheminezhad, M., Hashemi, S.M., Tahmasbi, M.: Ortho-radial drawings of graphs. Australas. J Comb. **44**, 171–182 (2009), `http://ajc.maths.uq.edu.au/pdf/44/ajc_v44_p171.pdf`

[72] Haskell, A.C., Breaznell, J.G.: Graphic charts in business: How to make and use them. Codex Book Company, Incorporated (1922)

[73] Hemetsberger, P.: Anagram generated by dict.cc/shuffle (2023), `https://m.dict.cc/shuffle/`. [Accessed on: 2023-12-31]

[74] Hliněný, P.: Crossing number is hard for cubic graphs. Journal of Combinatorial Theory, Series B **96**(4), 455–471 (2006)

[75] Hong, S.H., Merrick, D., do Nascimento, H.A.: Automatic visualisation of metro maps. Journal of Visual Languages & Computing **17**(3), 203–224 (2006)

[76] Huang, W.: Using eye tracking to investigate graph layout effects. In: 2007 6th International Asia-Pacific Symposium on Visualization. pp. 97–100. IEEE (2007)

[77] Huelsenbeck, J.P., Bollback, J.P., Levine, A.M.: Inferring the root of a phylogenetic tree. Systematic biology **51**(1), 32–43 (2002)

[78] IMAI, H., IRI, M.: Polygonal approximations of a curve — formulations and algorithms. In: TOUSSAINT, G.T. (ed.) Computational Morphology, Machine Intelligence and Pattern Recognition, vol. 6, pp. 71–86. North-Holland (1988). https://doi.org/https://doi.org/10.1016/B978-0-444-70467-2.50011-4, `https://www.sciencedirect.com/science/article/pii/B9780444704672500114`

[79] Inbar, O., Tractinsky, N., Meyer, J.: Minimalism in information visualization: attitudes towards maximizing the data-ink ratio. In: Proceedings of the 14th European conference on Cognitive ergonomics: invent! explore! pp. 185–188 (2007)

[80] Jacobsen, B., Wallinger, M., Kobourov, S., Nöllenburg, M.: Metrosets: Visualizing sets as metro maps. IEEE Transactions on Visualization and Computer Graphics **27**(2), 1257–1267 (2020)

[81] Jünger, M., Mutzel, P.: 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. In: Graph Algorithms and Applications I, pp. 3–27. World Scientific (2002)

[82] Kant, G.: Drawing planar graphs using the canonical ordering. Algorithmica **16**, 4–32 (1996)

[83] Katheder, J., Kobourov, S.G., Kuckuk, A., Pfister, M., Zink, J.: Simultaneous drawing of layered trees. In: International Conference and Workshops on Algorithms and Computation. pp. 47–61. Springer (2024)

[84] Kaufmann, M., Ueckerdt, T.: The density of fan-planar graphs. arXiv preprint arXiv:1403.6184 (2014)

[85] Kaufmann, M., Wagner, D.: Drawing graphs: methods and models. Springer (2003)

[86] Lawler, E.: A comment on minimum feedback arc sets. IEEE Transactions on Circuit Theory **11**(2), 296–297 (1964)

[87] Lazard, S., Lenhart, W.J., Liotta, G.: On the edge-length ratio of outerplanar graphs. Theoretical Computer Science **770**, 88–94 (2019)

[88] Leach, G.: Improving worst-case optimal delaunay triangulation algorithms. In: 4th Canadian Conference on Computational Geometry. vol. 2, p. 15. Citeseer (1992)

[89] Lempel, A.: An algorithm for planarity testing of graphs. In: Theory of Graphs: International Symposium. pp. 215–232. Gorden and Breach (1967)

[90] Limited, M.C.: Hong kong: Metro map (2024), `https://www.mtr.com.hk/en/customer/services/system_map.html`. [Accessed on: 2023-12-31]

[91] for London, T.: London: Tubemap (2023), `https://tfl.gov.uk/maps/track/tube`. [Accessed on: 2023-12-31]

[92] Masuda, S., Nakajima, K., Kashiwabara, T., Fujisawa, T.: Crossing minimization in linear embeddings of graphs. IEEE Transactions on Computers **39**(1), 124–127 (1990)

[93] Miller, Z., Orlin, J.B.: Np-completeness for minimizing maximum edge length in grid embeddings. Journal of algorithms **6**(1), 10–16 (1985)

[94] Modarres, M., Mirsamadi, M., Peyman, G.A.: Prevalence of congenital color deficiencies in secondary-school students in tehran. International ophthalmology **20**, 221–222 (1996)

[95] Muñoz, X., Unger, W., Vrt'o, I.: One sided crossing minimization is np-hard for sparse graphs. In: Graph Drawing: 9th International Symposium, GD 2001 Vienna, Austria, September 23–26, 2001 Revised Papers 9. pp. 115–123. Springer (2002)

[96] Niedermann, B., Rutter, I.: An integer-linear program for bend-minimization in ortho-radial drawings. In: Auber, D., Valtr, P. (eds.) Graph Drawing and Network Visualization - 28th International Symposium, GD 2020, Vancouver, BC, Canada, September 16-18, 2020, Revised Selected Papers. Lecture Notes in Computer Science, vol. 12590, pp. 235–249. Springer (2020). https://doi.org/10.1007/978-3-030-68766-3_19, `https://doi.org/10.1007/978-3-030-68766-3_19`

[97] Niedermann, B., Rutter, I., Wolf, M.: Efficient algorithms for ortho-radial graph drawing. arXiv preprint arXiv:1903.05048 (2019)

[98] Niedermann, B., Rutter, I., Wolf, M.: Efficient algorithms for ortho-radial graph drawing. In: Barequet, G., Wang, Y. (eds.) 35th International Symposium on Computational Geometry, SoCG 2019, June 18-21, 2019, Portland, Oregon, USA. LIPIcs, vol. 129, pp. 53:1–53:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019). https://doi.org/10.4230/LIPIcs.SoCG.2019.53, `https://doi.org/10.4230/LIPIcs.SoCG.2019.53`

[99] Nishizeki, T., Rahman, M.S.: Planar graph drawing, vol. 12. World Scientific (2004)

[100] Nöllenburg, M.: Automated drawing of metro maps. Citeseer (2005)

[101] Nöllenburg, M., Wolff, A.: Drawing and labeling high-quality metro maps by mixed-integer programming. IEEE Trans. Vis. Comput. Graph. **17**(5), 626–641 (2011). https://doi.org/10.1109/TVCG.2010.81, `https://doi.org/10.1109/TVCG.2010.81`

[102] Ohrhallinger, S., Mudur, S., Wimmer, M.: Minimizing edge length to connect sparsely sampled unstructured point sets. Computers & graphics **37**(6), 645–658 (2013)

[103] Page, R.D.: Tree view: an application to display phylogenetic trees on personal computers. Bioinformatics **12**(4), 357–358 (1996)

[104] Papadimitriou, C.H.: The np-completeness of the bandwidth minimization problem. Computing **16**(3), 263–270 (1976)

[105] Papakostas, A., Tollis, I.G.: Algorithms for area-efficient orthogonal drawings. Computational Geometry **9**(1-2), 83–110 (1998)

[106] Purchase, H.C.: Which aesthetic has the greatest effect on human understanding? In: Battista, G.D. (ed.) Graph Drawing, 5th International Symposium, GD '97, Rome, Italy, September 18-20, 1997, Proceedings. Lecture Notes in Computer

Science, vol. 1353, pp. 248–261. Springer (1997). https://doi.org/10.1007/3-540-63938-1_67, `https://doi.org/10.1007/3-540-63938-1_67`

[107] Purchase, H.C., Cohen, R.F., James, M.I.: Validating graph drawing aesthetics. In: Brandenburg, F. (ed.) Graph Drawing, Symposium on Graph Drawing, GD '95, Passau, Germany, September 20-22, 1995, Proceedings. Lecture Notes in Computer Science, vol. 1027, pp. 435–446. Springer (1995). https://doi.org/10.1007/BFb0021827, `https://doi.org/10.1007/BFb0021827`

[108] Purchase, H.C., Hoggan, E., Görg, C.: How important is the "mental map"?–an empirical investigation of a dynamic graph layout algorithm. In: Graph Drawing: 14th International Symposium, GD 2006, Karlsruhe, Germany, September 18-20, 2006. Revised Papers 14. pp. 184–195. Springer (2007)

[109] Ramshaw, L., Tarjan, R.E.: On minimum-cost assignments in unbalanced bipartite graphs. HP Labs, Palo Alto, CA, USA, Tech. Rep. HPL-2012-40R1 **20** (2012)

[110] Revell, L.J., Schliep, K., Valderrama, E., Richardson, J.E.: Graphs in phylogenetic comparative analysis: Anscombe's quartet revisited. Methods in Ecology and Evolution **9**(10), 2145–2154 (2018)

[111] Ringel, G.: Ein sechsfarbenproblem auf der kugel. In: Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg. vol. 29, pp. 107–117. Springer (1965)

[112] Ringel, G.: Map color theorem, vol. 209. Springer Science & Business Media (2012)

[113] Roberts, M.J.: Tube map central (2022), `http://www.tubemapcentral.com`

[114] Roberts, M.J., Newton, E.J., Canals, M.: Radi(c)al departures. Information Design Journal (IDJ) **22**(2) (2016)

[115] Robinson, D.F., Foulds, L.R.: Comparison of phylogenetic trees. Mathematical biosciences **53**(1-2), 131–147 (1981)

[116] Romli, F.I., Rafie, A.S.M., Wiriadidjaja, S.: Conceptual product design methodology through functional analysis. Advanced Materials Research **834**, 1728–1731 (2014)

[117] Ruzzo, W.L., Snyder, L.: Minimum edge length planar embeddings of trees. In: VLSI Systems and Computations, pp. 119–123. Springer (1981)

[118] Schaefer, M.: Realizability of graphs and linkages. In: Thirty Essays on Geometric Graph Theory, pp. 461–482. Springer (2012)

[119] Schneck, T.: New Parameters for Beyond-Planar Graphs. Ph.D. thesis, Universität Tübingen (2020)

[120] Schnyder, W.: Embedding planar graphs on the grid. In: Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms. pp. 138–148 (1990)

[121] Schöttler, S., Yang, Y., Pfister, H., Bach, B.: Visualizing and interacting with geospatial networks: A survey and design space. In: Computer Graphics Forum. vol. 40, pp. 5–33. Wiley Online Library (2021)

[122] Scornavacca, C., Zickmann, F., Huson, D.H.: Tanglegrams for rooted phylogenetic trees and networks. Bioinformatics **27**(13), i248–i256 (2011)

[123] Scott, J.: Network analysis: A handbook. Sage Publications (1992)

[124] Specker, E., Fried, E.I., Rosenberg, R., Leder, H.: Associating with art: A network model of aesthetic effects. Collabra: Psychology **7**(1), 24085 (2021)

[125] Stott, J., Rodgers, P., Martinez-Ovando, J.C., Walker, S.G.: Automatic metro map layout using multicriteria optimization. IEEE Transactions on Visualization and Computer Graphics **17**(1), 101–114 (2010)

[126] Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Transactions on Systems, Man, and Cybernetics **11**(2), 109–125 (1981)

[127] Tamassia, R.: Handbook of graph drawing and visualization. CRC press (2013)

[128] Tamassia, R., Battista, G.D., Batini, C.: Automatic graph drawing and readability of diagrams. IEEE Trans. Syst. Man Cybern. **18**(1), 61–79 (1988). https://doi.org/10.1109/21.87055, `https://doi.org/10.1109/21.87055`

[129] Tanahashi, Y., Ma, K.L.: Design considerations for optimizing storyline visualizations. IEEE Transactions on Visualization and Computer Graphics **18**(12), 2679–2688 (2012)

[130] Thomas, J.J.: Illuminating the path:[the research and development agenda for visual analytics]. IEEE Computer Society (2005)

[131] Wandell, B.A., Dumoulin, S.O., Brewer, A.A.: Visual cortex in humans. Encyclopedia of neuroscience **10**, 251–257 (2009)

[132] Ware, C., Purchase, H.C., Colpoys, L., McGill, M.: Cognitive measurements of graph aesthetics. Inf. Vis. **1**(2), 103–110 (2002). https://doi.org/10.1057/palgrave.ivs.9500013, `https://doi.org/10.1057/palgrave.ivs.9500013`

[133] Weyl, H.: Symmetry, vol. 104. Princeton University Press (2015)

[134] Wickham, H., Cook, D., Hofmann, H., Buja, A.: Graphical inference for infovis. IEEE transactions on visualization and computer graphics **16**(6), 973–979 (2010)

[135] Wolf, M.: Bend Minimization of Ortho-Radial Graph Drawings. Ph.D. thesis, Informatics Institute (2016)

[136] Xu, Y., Chan, H., Chen, A.: Automated generation of concentric circles metro maps using mixed-integer optimization. Int. J. Geogr. Inf. Sci. **36**(12), 2386–2411 (2022). https://doi.org/10.1080/13658816.2022.2102636, `https://doi.org/10.1080/13658816.2022.2102636`

[137] Yau, S.S., Grabow, P.C.: A model for representing programs using hierarchical graphs. IEEE Transactions on Software Engineering (6), 556–574 (1981)

[138] Yourdon, E.: Structured programming and structured design as art forms. In: Proceedings of the May 19-22, 1975, national computer conference and exposition. pp. 277–277 (1975)