UNIVERSITÄT PASSAU

Fakultät für Informatik und Mathematik

Habilitation Thesis

A Generalized Framework for Drawing Directed Graphs

Christian Bachmaier

Faculty of Informatics and Mathematics
University of Passau

October 30, 2009

Cumulative habilitation thesis for the aquisition of the venia legendi in Computer Science from the Faculty of Informatics and Mathematics, University of Passau.

Directed Graphs are usually layouted with the hierarchic Sugiyama framework, which is indeed one of the most important drawing methods for graphs. It places the vertices on parallel level lines and attempts to map the directions of the edges to a uniform geometric direction, e. g., from top to bottom. Then the resulting drawing visualizes a common direction of information flow stored by the structure of the input graph.

In this thesis we generalize the traditional hierarchic drawing style in two ways: radial drawings with level lines forming concentric cycles and cyclic drawings with levels forming a star, i. e., recurrent hierarchies. Further, we allow edges between vertices within the same level, which often occur in practice. Our main results are a complete framework for both layout conventions and a major enhancement of the reduction of edge crossings using a new global optimization method. This approach also upgrades traditional horizontal level drawings.

Applications of (drawings of) general level graphs include for example state or flow charts, schedules, UML class diagrams, entity relationship diagrams, and biochemical pathways. Radial drawings are especially useful to visualize social or policy networks, i. e., to map structural centrality like importance or role of the vertices/actors on geometric centrality or simply to display the vicinity of a distinguished actor. Typical applications of cyclic level drawings are the visualizations of cyclic structures within graphs as they can be found for example in periodic event scheduling, biochemical pathways forming cycles or spirals, and replication of arrays of identical adjoining substructures in VLSI design, e. g., cells on memory modules. A combination of the radial and cyclic drawing styles gives interactive 2D views on the torus, which are infinitely scrollable in four directions. This is suitable for drawings which avoid long edges and their crossings while preserving the mental map, e. g., for UML class diagrams.

Preface

I wish to express my gratitude to all to whom I am indebted, directly or indirectly in writing this thesis. First of all, thanks are due to my supervisor, Professor Dr. Franz J. Brandenburg, who introduced me to graph drawing. Unlike me, he was sure even before I finished my diploma that I was capable of scientific work. Therefore, he was the one who encouraged me to have a "look" at hierarchical graphs. I am very grateful to Franz for supporting my work in every aspect I ever could imagine. He and later additionally the Faculty of Informatics and Mathematics of the University of Passau gave me the opportunity and freedom to take part in the research in computer science and to take part in conferences. It were the Graph Drawing symposia that have been very important for me.

Further, I am very grateful to my other scientific mentor Professor Dr. Ulrik Brandes. It was a joy to join his algorithmic group at the University of Constance in 2005 as full member and it was a very fruitful time for me. Over all the years, for every single of my numerous concerns and often difficult questions he continuously had the right solutions. He always had time for me on the spot although his tight schedule and never got tired of listening and answering. Last but not least, my sincere thanks go to all my co-authors for writing scientific articles together with me. Among them, I especially wish to thank Wolfgang Brunner and Dr. Michael Forster for intensive and always fruitful cooperation on various problems related to level graphs.

Christian Bachmaier

Contents

1 Introduction and Survey

Visually conditioned informations, e. g., from a metro map on Monday morning to an election forecast on Sunday evening, are ubiquitous. Most of the time humans interpret visualizations unconsciously the right way, but some illustrations are confusing and raise misunderstandings, and others are willfully misleading. Good visualizations impart more information in the blink of an eye and upon smallest area than simple text ever can describe.

Humans can percept most informations of all with the eyes [64]. This requires at all times new and more elaborate ways of visualization, since the information amount is growing extremely fast, in high contrast to the human perception and comprehension. With growing information floods it is getting more and more difficult to maintain the overview, to distinguish between major and minor facts, and to isolate the desired informations from all other.

A folklore example [98]: The city of London describes their metro map graphically since the end of the 19th century. This was done in a way as it is common for geographical plans. The visualization showed the ways with different distances for each passage and with a street map in the background for an easy orientation. With the growth of the train net it got more and more difficult to visualize it clearly arranged. In 1937 the draftsman Henry Beck has reduced the visual complexity of the map radically. He has straighten out the lines and has allowed angles of 45 or 90 degrees only. By the deliberate omittance of unnecessary details, the map was concentrated to the major facts. From now on the users had no orientation in a combination with the street map, no basis about the real distances between the stations, and no information about the exact routing of the tracks any more. However, for using the plan it was enough to know where to enter or exit the trains. The reduced map was so successful, that this principle was copied world wide.

Nowadays, the flood of informations to visualize – usually generated with the help of machines – is in nearly all cases far too much for manual editing. Furthermore, manual layout is a time consuming exercise with relatively poor results for all but the smallest of visualizations. Thus, there has been considerable interest in automatic or at least semi-automatic drawing algorithms. These are made possible since in the mid 1980s graphics workstations became standard. However, algorithms need a well defined input format, i. e., there is the need to abstract from (relational) information and convert it to the mathematical model of a graph with vertices for the entities and edges for relations between them. This is the point where graph drawing comes into play. Here, the goal is to layout the information structure, i. e., the corresponding graph, such that it is easy to understand and to remember. The usefulness of a visualization depends on the layout of the underlying graph: a "nice" drawing can be a great aid to a designer. The computer science pioneer Donald E. Knuth stated at the Graph Drawing Symposium 1996: "Graph drawing is the best possible field I can think of. It merges aesthetics, mathematical beauty, and wonderful algorithms."

The framework for hierarchical layouting [138], often named after one of its inventors the Sugiyama algorithm, is nowadays the most common method for drawing directed graphs. By preserving an uniform edge direction the resulting layouts visualize a common direction of information flow, e. g., for the illustration of chronological or casual dependencies. This maps topological or structural direction to a geometrical direction. Instead of a monolithic algorithm the authors of [138] describe a modular framework which features four consecutive phases. The phases are self-contained and thus exchangeable, which provides a high flexibility of processing. In our opinion, this is the deeper reason for the framework's success. The particular phases are *cycle removal*, *level assignment*, *crossing reduction*, and *coordinate assignment*. Unfortunately, almost all problems which occur during the single phases of this approach are $\mathcal{NP}$-hard. Nevertheless, for all these problems appropriate heuristics have been developed and nowadays nearly all general graph drawing software, e. g., [8, 32, 61, 79, 118, 146], use this approach, mostly enriched by modifications required in practice like vertices of different sizes and shapes, ordering constraints, clustering, etc.

Our goal is to extend the framework such that it generates drawings which do not only use horizontal level lines for visualizing a top-down information flow. In fact, we use concentric level lines for a flow from a center to the outside and we use levels forming a star to highlight cyclic dependencies.

Radial drawings are the most important and well-known drawing conventions in social network analysis and visualization, in particular, displaying *centrality* indices (importance) of actors [21, 144]. Nowadays, the practical relevance of radial drawings can be seen on the exploding popularity of social network and contact platforms in the internet like Facebook, Google's Orkut, MySpace, Xing, or LinkedIn, to name only a few. A local player in Germany, studiVZ, which started in late 2005 had in October 2007 more page impressions than the major local sites T-Online, mobile.de and Yahoo Germany in sum [145]. On the other hand, cyclic drawings are especially useful for an easy perception of cycles. For example, there are well-known cycles in the biosciences, where it is a common standard to display these cycles as such, e. g., the citrate cycle or the fatty acid synthesis. These cycles often serve as a landmark [110] and should be visualized in their natural circular shape. Another inevitable use are repeating processes, such as daily, weekly, or monthly schedules with the same tasks, which define the Periodic Event Scheduling Problem [132]. Again, it is important that these cycles are clearly visible; this makes the drawing "nice".

1.1 Preliminaries

We first recall some basic concepts and notions of graphs as they are commonly used, e. g., in [35, 37, 47, 119]. The established visualization methods for different classes of graphs are summarized in [42, 96, 136], which cover the basics of graph drawing.

1.1.1 Graphs

A *graph* $G = (V, E)$ is a finite non-empty set of *vertices* V which are connected by *edges* E. It is *directed* (a *digraph*) if the edges are ordered pairs of vertices and *undirected* if the order does not matter. Thus, we denote a directed edge by $e = (u, v)$ and an undirected edge by $e = \{u, v\}$. The vertices u and v are the *end vertices* of an edge $e = \{u, v\}$ and are called *adjacent* to each other. Then it is said that u and v are *neighbors*. An end vertex is called *incident* to its edge(s) and vice versa. A vertex is *isolated* if it has no incident edges. Consider two directed edges (u, v) and (v, w). Then (u, v) is called an *incoming edge* of v and (v, w) is called *outgoing edge* of v. The *degree* $\deg(v)$ of a vertex v is the number of incident edges. Let $\deg(G) := \max\{\deg(v)|v \in V\}$ be the *degree of the graph*. Let $N^-(v) := \{\, u \mid (u, v) \in E \,\}$, $N^+(v) := \{\, v \mid (v, w) \in E \,\}$, and $N(v) := N^-(v) \cup N^+(v)$ denote the neighborhood of a vertex v. Similarly denote by $E^-(v) := \{(u, v) \in E\}$, $E^+(v) := \{(v, w) \in E\}$, and $E(v) := E^-(v) \cup E^+(v)$ the sets of incident edges. A graph is *complete* if it contains all possible edges. The *density* of a graph is the ratio of its number of edges over the number of edges of the corresponding complete graph.

Usually graphs are visualized in such a way that vertices are drawn as points, circles, or squares and edges are displayed as curves between vertices. For a digraph an edge $e = (u, v)$ is drawn as an arrow from its *source vertex* u to its *target vertex* v. For an undirected graph an edge is drawn as a simple line between its end vertices. There is no arrow, indicating the symmetry.

Parallel edges are two or more edges with the same end vertices. A *reflex edge* is an edge (v, v). A *simple graph* does not contain parallel or reflex edges. A *multi graph* is a graph without reflex edges. In the following we use simple graphs for simplicity only. However, all presented concepts and algorithms can deal with multi graphs out of the box or can easily be extended.

A graph $G^* = (V^*, E^*)$ is called a *subgraph* of the graph $G = (V, E)$ if $V^* \subseteq V$ and $E^* = \{\, (u, v) \mid u, v \in V^* \,\} \subseteq E$ hold.

A *(directed) path* $P = (v_1, \ldots, v_p)$ is a sequence of vertices v_i with $(v_i, v_{i+1}) \in E$ for $1 \le i \le p-1$. P is *simple* if all vertices are mutually distinct. P is a *cycle* if $v_1 = v_p$ for $p > 2$. P is a *simple cycle* if all vertices except v_1 and v_p are mutually distinct. A directed graph is *strongly connected* if there exists a directed path between every pair of its vertices. A maximal subgraph with such a property is called a *strongly connected component* (SCC).

Every graph $G = (V, E)$ can be traversed by a *breadth first search* (BFS) using a queue or by a *depth first search* (DFS) using a stack or recursion in $\mathcal{O}(|V| + |E|)$ time.

1.1.2 DAGs

A DAG is a *directed acyclic graph*, i. e., it contains no cycles. In a DAG, a vertex with no incoming edges is called a *source* and a vertex with no outgoing edges is called a *sink*. Each DAG $G = (V, E)$ has a *topological sorting* of its vertices, which is a linear ordering of all its vertices $v \in V$ such that if E contains a directed edge (u, v) then u appears before v in this ordering. Such a vertex ordering is not unique and can be found efficiently in $\mathcal{O}(|V| + |E|)$ time by successively removing all sources.

1.1.3 Level Graphs

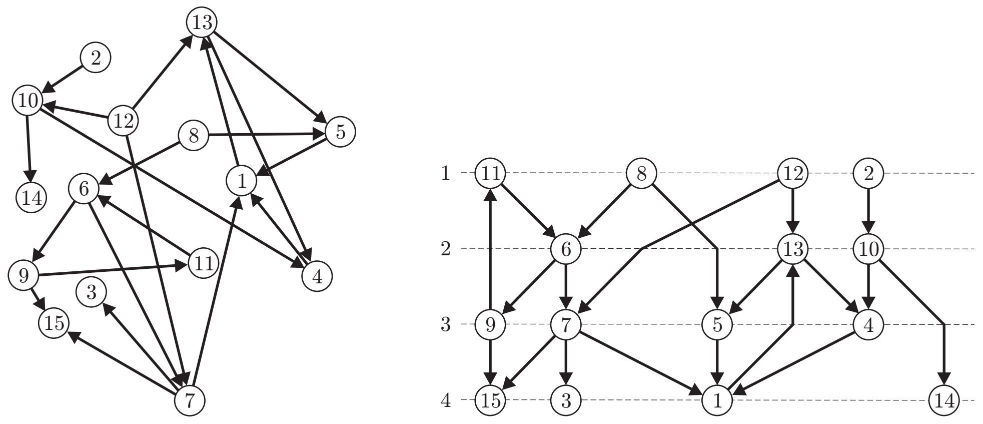
Now we specialize to k-level graphs, which is one of the central notions of this thesis. A *k-level graph* $G = (V, E, \phi)$ with $k \leq |V|$ is a *k-partite* graph with a surjective level assignment $\phi\colon V \to \{1, 2, \ldots, k\}$ that partitions the vertex set into k pairwise disjoint subsets $V = V_1 \dot\cup V_2 \dot\cup \ldots \dot\cup V_k$, $V_j = \phi^{-1}(j)$, $1 \leq j \leq k$, such that $\phi(u) \neq \phi(v)$ for each edge $(u, v) \in E$. A k-level graph is *proper* if each edge $e = (u, v) \in E$ is *short*, i. e., $\mathrm{span}(e) := |\phi(u) - \phi(v)| = 1$. Otherwise, it has a *long edge* which spans several levels. Each level graph can be made proper by subdividing long edges with additional dummy vertices, i. e., replacing it with a path of proper edges. In total up to $\mathcal{O}(k \cdot |E|)$ dummy vertices may be needed. We call a proper edge incident to two dummy vertices an *inner segment*. All other proper edges are called *outer segments*. For proper level graphs let $E_i := \{\, (u, v) \mid u \in V_{i-1}, v \in V_i, 1 < i \leq k \,\} \subseteq E$ be the edges between levels $i - 1$ and i. In our examples we draw dummy vertices either as small black circles or as bends. Level graphs are a generalization of *bipartite graphs* which have only two levels and no edges between the vertices within the same level.

A *(level) embedding* $\mathcal{E}$ of a proper level graph is a partial order $\prec$ of the vertices V such that $u \prec v$ or $v \prec u$ iff $\phi(u) = \phi(v)$.[1] Define the (not necessarily consecutive) positions of the vertices as a function $\pi\colon V \to \mathbb{Z}$ with $u \prec v \Leftrightarrow \pi(u) < \pi(v)$ for any two vertices $u, v \in V_i$ on the same level i. Let the *crossing function* $\chi_{\mathcal{E}} : E \times E \to \{0, 1\}$ be 1 for mutually distinct edges $e_1, e_2 \in E$ if e_1 and e_2 cross in the embedding $\mathcal{E}$ and 0 otherwise.

[1]For non-proper level graphs the given definition of $\mathcal{E}$ is not sufficient: It does not store between which pairs of consecutive vertices long edges are routed on spanned levels. Hence, non-proper embeddings additionally consist of cyclic orderings of the incident edges for every vertex. Because of isolated vertices, we cannot (completely) omit the vertex orderings.

1.2 Sugiyama Algorithm

The display of hierarchical structures is one of the key topics in automatic graph drawing [34]. To visualize a common direction of information flow, directed graphs are usually drawn such that the vertices are placed on parallel horizontal levels, and the edges are straight lines or y-monotone polylines or splines.



(a) A digraph

(b) A level drawing of (a)

Figure 1.1. A directed graph and a level drawing

This technique is used by the Sugiyama[2] algorithm, the most common algorithm for drawing DAGs, level graphs, and arbitrary directed graphs [42, 56, 96, 138]. The algorithm operates in four consecutive phases which we will describe together with their state of the art implementations next. In each phase, the goal is to assure some aesthetic and readability criteria as good as possible. The following list shows some widely accepted and important criteria [96] but maybe enlarged by individual preferences:

1. Edges pointing upward should be avoided.
2. Vertices should be evenly distributed.
3. Long edges should be avoided.
4. Edges should be as straight/vertical as possible.
5. Edge crossings should be avoided.
6. Area consumption should be low.
7. Aspect ratio should be reasonable.

[2]Although there was some initial work on level drawings by Warfield [143] and Carpano [31] before, this approach is commonly attributed to Sugiyama et al. [138].

1.2.1 Cycle Removal

In the first phase, called *cycle removal*, the directed input graph is made acyclic by reversing the direction of appropriate edges. Reversing a minimum set of edges is known as the *feedback arc set problem* and is $\mathcal{NP}$-hard [75, 95]. Clearly, the goal is to keep the *feedback set* (FAS), i. e., the set of reversed edges, as small as possible. This keeps the changes in the graph small. At the end of the algorithm the reversed edges are reversed again to obtain their initial orientation.

A simple heuristic is to choose an arbitrary ordering of the vertices and reverse all edges from a higher order vertex to a lower order vertex. Another simple and fast greedy strategy is to successively reverse for every vertex either the set of incoming edges or all outgoing ones. Taking always the smaller set leads to $|\text{FAS}| \leq \frac{|E|}{2}$. The enhanced greedy heuristic of Eades et al. [53] is basically the same but proceeds the vertices in a special order. After hiding all sources and sinks, which never can be part of any cycle, the next vertex $v \in V$ in the loop is always the one with minimal value of $|N^+(v)| - |N^-(v)|$. They show that this results in an upper bound of $|\text{FAS}| \leq \frac{|E|}{2} - \frac{|V|}{6}$ and that a computation needs only $\mathcal{O}(|E|)$ time. Sander [131] suggests a more elaborate but similar version of the above and chooses the next vertex v on the basis of strongly connected components. He reports promising practical results, however, the same theoretical bounds and a running time in $\mathcal{O}(|V| \cdot |E|)$. Berger and Shor [18] presented a randomized greedy heuristic with $|FAS| \leq \frac{1}{2} + \Omega\left(\frac{1}{\sqrt{\deg(G)}}\right) \cdot |E|$. As yet, the best known approximation factor is $\mathcal{O}(\log |V| \cdot \log \log |V|)$ by an algorithm of Even et al. [63]. The authors of [39–41] measure their approximation quality against the longest cycle in the graph. Rowe et al. [126] and Gansner et al. [73] suggest to take DFS or the appearance of edges in cylces into account. Saab [128] proposes a divide & conquer strategy. A formulation as an integer linear program (ILP) for computing the exact solution can be found in [78].

1.2.2 Level Assignment

During the second phase, called *level assignment*, the vertices are assigned to horizontal levels. Thereby, minimizing both the *height*, i. e., the number of levels k, and the *width* ω, i. e., the maximum number of vertices per level, is $\mathcal{NP}$-hard as a straightforward reduction from the precedence constraint multiprocessor scheduling problem shows [36, 75]. Nevertheless, the goal is to get compact drawings with a good aspect ratio.

Minimizing the Height Computing a leveling with minimum height can be done by applying the *longest path method*: First, each source of the graph is assigned to level 1. For the remaining vertices v the levels are recursively defined by $\phi(v) = \max\{\, \phi(u) \mid (u, v) \in E \,\} + 1$. This puts each vertex on the lowest possible level while minimizing the number of levels k. There is no explicit balancing of level sizes, however. The algorithm can be implemented in linear time $\mathcal{O}(|V| + |E|)$ by using a topological ordering of the vertices.

Given Width For a better vertex distribution consider the Coffman/Graham algorithm [36], which explicitly maintains a given maximum width ω (ignoring edges). The first step of the algorithm is as follows: After removing transitive edges in linear time, an appropriate numbering $o\colon V \to \{0, \dots, |V|-1\}$ of the vertices is computed. Initially, all vertices are unnumbered. We consecutively choose one vertex at a time and assign the next ascending number to it. We choose the vertex such that it has no unnumbered successors and that the numbers of the successors are minimal regarding a specific ordering of integer sets. A set of vertex numbers is considered less than another one, if the maximum is less. If the maximum of both sets is equal, the next smaller value is compared, and so on. In the second phase of the algorithm we place one vertex at a time, starting with the vertex numbered with $|V|-1$ on level $i = 1$ and filling the levels from lower to higher numbers. In one step we place the next unleveled vertex $v \in V$ with maximal $o(v)$ whose predecessors are already leveled. If level i is full, i. e., if it already contains $|V_i| = \omega$ vertices, or if v has a predecessor u with $\phi(u) = i$, then we start a new level, i. e., we increase i by 1. We then set of v to $\phi(v) = i$. The whole algorithm can be implemented to run in $\mathcal{O}(|V| + |E|)$ time [102].

Lam and Sethi [101] have shown that the number of levels k of the computed leveling with width ω is bounded by $k \leq (2 - \frac{2}{\omega}) \cdot k_{\text{opt}}$, where k_{opt} is the minimum height of all levelings with width ω. Thus, the Coffman/Graham algorithm is an exact algorithm for $\omega \leq 2$. It is currently the most commonly used leveling method. However, in its process it does not consider the emerging dummy vertices in the width, which is also $\mathcal{NP}$-hard [27]. Healy and Nikolov [84] presented an ILP approach for the computation of levelings with given width and height while counting both original and dummy vertices for the width.

Minimizing the Total Edge Span Before the third phase starts, traditionally the graph is made proper. Long edges between vertices of non-adjacent levels are replaced by chains of dummy vertices and proper edges between the corresponding adjacent levels as a *normalization*. The dummy vertices represent potential edge *bends* and the routing of long edges among the vertices on the spanned levels.

Minimizing the total edge span in (1.1) is equivalent to minimizing the number of dummy vertices. This also minimizes the height [56]. The ILP can be solved in polynomial time because of a totally unimodular constraint matrix [96].

$$
\begin{aligned}
&\operatorname{span}(G) = \min \sum_{(u,v)\in E} (\phi(v) - \phi(u)) \\
&\text{subject to} \\
&\qquad \begin{array}{lll} \phi(v) - \phi(u) \geq 1 & \text{for} & (u,v) \in E \\ \phi(v) \in \mathbb{N} & \text{for} & v \in V \end{array}
\end{aligned}
\tag{1.1}
$$

The complexity of algorithms in the Sugiyama framework heavily depends on the number of dummy vertices inserted. Although this number can be minimized efficiently, it may still be in the order of $\mathcal{O}(|V| \cdot |E|)$ [69].

1.2.3 Crossing Reduction

In the third phase, called *crossing reduction*, orderings of the vertices within the levels are computed such that the number of edge crossings is reduced or even minimized. Clearly, two edges $e_1 = (u, v), e_2 = (x, y) \in E$ between two levels are crossing if and only if $(\pi(x) - \pi(u)) \cdot (\pi(y) - \pi(v)) < 0$. Empirical studies by Purchase [122, 123] have shown, that a low number of crossings is one of the major factors for supporting an easy human perception of graphs.

However, crossing minimization is $\mathcal{NP}$-hard [76], even if there are only two levels [93] and the vertices of one level are fixed [55, 57, 58]. This is also true for sparse graphs with vertex degree 4 [112]. The *one-sided two-level crossing minimization problem* is fundamental in graph drawing and has received great attention in literature [42]. Therefore, the traditional Sugiyama algorithm uses one (or more) of the many and intensively investigated heuristics and performs it successively level by level in subsequent top-down and bottom-up *sweeps*. These are repeated a constant number of times or until no further reduction of crossings can be achieved. In the best case no crossings remain at all and the graph is drawn *level planar*. However, a heuristic does not guarantee a planar drawing even if one exists, although in this case it might be especially desirable to avoid crossings. Fortunately, there are efficient algorithms for testing this property and for constructing a drawing without a crossing [80, 83, 91, 124].

Experiments by Jünger and Mutzel [92] and Matuszewski et al. [108] showed that the results of the level-by-level sweep are far from optimum. Bastert and Matuszewski [96, page 102] claim that "one can expect better results by considering all levels simultaneously", but also state that "k-*level crossing minimization* is a very hard problem".

1.2.3.1 One-Sided Crossing Reduction

Since the repeated one-sided crossing reductions are state of the art for reducing crossings in level graphs, we describe the most popular ones for levels $i - 1$ (fixed) and $1 < i \leq k$ (permutable) for a proper graph k-level $G = (V, E, \phi)$ next. For a survey see [96].

Barycenter and Median Heuristics The fast and simple *barycenter* and *median heuristics* are based on the intuition that if each vertex is close to its neighbors, then the edges tend to be short and thus generate fewer crossings than long edges. The vertices in the fixed level $i - 1$ are numbered from 1 to $|V_{i-1}|$. The heuristics assign to each vertex $v \in V_i$ in the permutable level i the barycenter value resp. median position of its predecessors in V_{i-1}. After that V_i is sorted according to these values. Both heuristics avoid type 2 conflicts, i. e., crossings of two edges each connecting two dummy vertices. The running times for one level stay in $\mathcal{O}(|E_i| + |V_i| \log |V_i|)$ if computing the median value of each vertex $v \in V_i$ in $\mathcal{O}(|N^-(v)|)$ time [37]. The $\log$-factor can be eliminated for the integral vertex positions of the median heuristic by using bucket sort.

For the one-sided two-level crossing reduction, the barycenter heuristic is an $\mathcal{O}(\sqrt{n})$-approximation to the optimum [58] and it gives a drawing without crossings if one is possible. The median heuristic is a 3-approximation [58].

Sifting Heuristic The *sifting heuristic* has a higher runtime than the above simple heuristics, however, it produces fewer crossings in practice [77]. Sifting was originally introduced as a heuristic for the minimization of *ordered binary decision diagrams* [127], i. e., optimizing the orderings of the variables, that are widely used to model Boolean functions in formal logic verification and synthesis. Later it was adapted to the one-sided crossing minimization problem [108]. The main idea is to keep track of the objective function while in a *sifting step* a *traversing vertex* $u \in V_i$ is moved along a fixed ordering of all other vertices in V_i. Then u is placed at its locally optimal position. This is done by iteratively swapping consecutive vertices only.
The method is thus an extension of the *greedy-switch heuristic* [52], where u after putting it to the front is swapped iteratively with its successor. We call a single swap a *sifting swap*. Executing a sifting step for every vertex in V_i is called a *sifting round*. For crossing reduction, the objective function is the number of crossings between the edges incident to the vertex under consideration and all other edges. The efficient computation of the crossing count in sifting is based on the *crossing matrix*. The $|V_i|^2$ entries in the crossing matrix χ_{uv} as formally defined by (1.2) correspond to the number of crossings caused by (the edges of) pairs of vertices $u \prec v \in V_i$ in a particular relative ordering in embedding $\mathcal{E}$ and can be computed as a preprocessing step in $\mathcal{O}(|E_i|^2)$ time [141, 147]. Whenever a vertex is placed at a new position, only a small number of updates is necessary [77].

$$\chi_{uv} = \sum_{s \in N^-(u)} \sum_{t \in N^-(v)} \chi_{\mathcal{E}}\left((s,u),(t,v)\right) \tag{1.2}$$

Let $\mathcal{E}$ and $\mathcal{E}'$ be two embeddings of $G^* = (V_{i-1} \dot\cup V_i, E_i, \phi) \subseteq G$ $(1 < i \leq k)$, where $\mathcal{E}'$ is computed from $\mathcal{E}$ by swapping the vertex $u \in V_i$ and its successor $v \in V_i$. Since swapping positions of u and v only affects crossings of incident edges, the number of crossings in $\mathcal{E}'$ can be efficiently computed by (1.3).

$$\chi_{\mathcal{E}'} = \chi_{\mathcal{E}} + \underbrace{\chi_{vu} - \chi_{uv}}_{=:\Delta} \tag{1.3}$$

This allows a running time in $\mathcal{O}(|V_i|^2)$ for one round. In practice, only few sifting rounds (3–5 for typical problem instances) are necessary to reach a local optimum for all vertices simultaneously. According to our experiments it seems that this is in most cases also the global optimum which we have computed for small graphs with the ILP formulation of [92]. The largest reduction of crossings usually occurs in the first round. In practice sifting obtains a considerable speed-up by considering lower bounds during the algorithm, allowing to prune parts of the search space [77]. However, the theoretical time complexity is unaffected by this improvement.

Exact Crossing Minimization Currently, the best known approximation algorithm for the one-sided crossing minimization problem by Nagamochi [116] gives a 1.4664-approximation. This multiplicative upper bound does even hold for the well known lower bound of the crossing number in (1.4), where χ_{uv} are the entries of the crossing matrix.

$$\chi_{\text{lower}} = \sum_{\{u,v\}\subseteq V_i} \min\{\chi_{uv}, \chi_{vu}\} \tag{1.4}$$

Valls et al. [141] and Jünger and Mutzel [92] presented the following ILP for computing the optimal level permutations to obtain the minimum number of crossings. For simplicity we here identify the vertex names with a numbering from 1 to $|V|$. For both levels $l \in \{i-1, i\}$ and all vertices $u, v \in V_l$ define $\delta^l_{uv} = 1$ if $u \prec v$ and $\delta^l_{uv} = 0$, otherwise. Then we first compute the crossing matrix with entries χ_{uv}.

$$\chi_{uv} = \sum_{s\in N^-(u)} \sum_{t\in N^-(v)} \delta^{i-1}_{ts} \tag{1.5}$$

The variables $x_{uv} = \delta^i_{uv}$ in (1.6) store after solving (1.6) the desired relative vertex positions on level i, i. e., $u \prec v \Leftrightarrow x_{uv} = 1$.

$$\begin{aligned} \chi'_{\text{lower}} = \min \sum_{u=1}^{|V_i|-1} \sum_{v=u+1}^{|V_i|} (\chi_{uv} - \chi_{vu}) \cdot x_{uv} \\ \text{subject to} \\ 0 \le x_{uv} + x_{vw} - x_{uw} \le 1 \quad &\text{for} \quad u, v, w \in V_i, \\ &\qquad 1 \le u < v < w \le |V_i| \\ x_{uv} \in \{0, 1\} \quad &\text{for} \quad u, v \in V_i, \\ &\qquad 1 \le u < v \le |V_i| \end{aligned} \tag{1.6}$$

The optimal crossing number can then easily be computed with (1.7). According to our experiments in JAVA using the library lp_solve [19], the approach can be applied to graphs with up to 150 vertices in practice.

$$\chi_{\text{opt}} = \chi'_{\text{lower}} + \sum_{u=1}^{|V_i|-1} \sum_{v=u+1}^{|V_i|} \chi_{vu} \tag{1.7}$$

It is somewhat surprising that using the barycenter heuristic instead of an exact algorithm for iterated one-sided two-level crossing reduction within the sweeping approach over k levels yields slightly better results with fewer crossings [62, 92]. So, for more than two levels, the exact level-by-level approach seems not to be the satisfying solution.

1.2.3.2 k-level Crossing Reduction

Up to now there are only few k-level crossing reductions which consider all levels simultaneously. Each of them has some drawbacks. Similar to the iterative sweeps above, they all need proper level graphs $G = (V, E, \phi)$ as input.

Ordered k-Level Sifting A slightly more global approach using sifting has been introduced by Matuszewski et al. [108], which we call *ordered k-level sifting* to avoid confusion with our approach described later. In a sifting round, they perform a sifting step for each vertex of the graph in degree decreasing order. If a round does not improve the number of crossings, the vertex order is reversed for increasing degree. Thus, the heuristic does not sweep level-by-level but is still limited to a local view as long edges are not treated as a whole. Contrary to one-sided crossing reductions, here all edges incident to the swap vertices, i. e., incoming and outgoing edges, are considered for computing the change in the number of crossings Δ. Thus, the approach rather corresponds to a *centered three-level crossing reduction*, i. e., treating three consecutive levels V_{i-1}, V_i, V_{i+1} and permuting V_i while the orders of V_{i-1} and V_{i+1} are fixed such that the crossings between the three levels are reduced. The theoretical time bounds are in both cases the same as for the sweeping sifting. However, because of the local view for reducing crossings, it is likely that the approach gets stuck in local optima.

Tutte's Algorithm Tutte's algorithm [56] is similar to barycenter. After fixing the permutation of levels 1 and k, in each other level the x-coordinate of a vertex v is chosen as the weighted average of its neighbors.

$$x(v) = \frac{1}{2 \cdot |N^-(v)|} + \sum_{u \in N^-(v)} x(u) + \frac{1}{2 \cdot |N^+(v)|} + \sum_{w \in N^+(v)} x(w) \tag{1.8}$$

This results in a system of sparse linear equation, which to solve may need exponential time, however. In the last step the levels are sorted according the x-values.

Exact Crossing Minimization Jünger et al. [90] present an ILP formulation for computing the minimum crossing number for a k-level graph. For simplicity of its description we identify the vertex names with a numbering from 1 to $|V|$. For levels $1 < i \leq k$ and all pairs of edges $(u, v), (x, y) \in E_i$ we define $c^i_{uvxy} = 1$ if (u, v) and (x, y) cross and $c^i_{uvxy} = 0$, otherwise. The variables x^i_{ux} in (1.9) store after solving it the desired relative vertex positions on level $1 \leq i \leq k$. A value of $x^i_{ux} = 1$ means $u \prec x$ and a value of $x^i_{ux} = 0$ means $u \succ x$.

$$\begin{aligned} \chi_{\text{opt}} &= \min \sum_{i=2}^{k} \sum_{(u,v),(w,y) \in E_i, 1 \leq u < w \leq |V_{i-1}|, 1 \leq v < y \leq |V_i|} c^i_{uvwy} \\ &\text{subject to} \\ &-c^i_{uvwy} \leq x^i_{vy} - x^{i-1}_{uw} \leq c^i_{uvwy} \quad \text{for} \quad \begin{array}{l} (u,v),(w,y) \in E_i, \\ 1 \leq u < w \leq |V_{i-1}|, \\ 1 \leq v < y \leq |V_i|, \\ 1 < i \leq k \end{array} \end{aligned} \tag{1.9}$$

$$
\begin{array}{lll}
1 - c^i_{uvwy} \le x^i_{yv} - x^{i-1}_{uw} \le 1 + c^i_{uvwy} & \text{for} & (u,v),(w,y) \in E_i, \\
 & & 1 \le u < w \le |V_{i-1}|, \\
 & & 1 \le y < v \le |V_i|, \\
 & & 1 < i \le k \\
0 \le x^i_{uw} + x^i_{wz} - x^i_{uz} \le 1 & \text{for} & u,w,z \in V_i, \\
 & & 1 \le u < w < z \le |V_i|, \\
 & & 1 \le i \le k \\
x^i_{uw} \in \{0,1\} & \text{for} & u,w \in V_i, \\
 & & 1 \le u < w \le |V_i|, \\
 & & 1 \le i \le k \\
c^i_{uvwy} \in \{0,1\} & \text{for} & (u,v),(w,y) \in E_i, \\
 & & 1 \le u < w \le |V_{i-1}|, \\
 & & 1 \le v < y \le |V_i|, \\
 & & 1 < i \le k
\end{array}
$$

Jünger et al. state that their branch & cut approach is only practicable if additional and deeper polyhedral studies are conducted for speedup as already done so far in [90] (with the introduction of some additional constraint inequalities/cutting planes on the associated polytope). According to our implementation within the Gravisto framework [8] using the ILP solver of [19], the approach can be applied to graphs with up to 40 vertices in practice. The concrete running times heavily depend on the uniform edge distributions among the levels.

Metaheuristics To attack the global k-level crossing minimization problem also metaheuristics have been proposed in literature, such as genetic algorithms [99, 140], tabu search [100], or windows optimization [62]. These general (stochastic) global search approaches usually compute good solutions with few crossings at the expense of high running times.

1.2.4 Coordinate Assignment

The fourth phase, called *coordinate assignment*, computes an x-coordinate $x(v)$ for every vertex $v \in V$. It is usually constrained to preserve the ordering determined in the third phase and to introduce a minimum separation space δ between vertices within a level. The y-coordinates are given by the levels. Finally, the dummy vertices introduced by the normalization are removed and replaced by edge bends. In the following let $G' = (V', E', \phi)$ be the normalized version of the k-level graph $G = (V, E, \phi)$.

Straight-Line Every embedding of a k-level graph has a straight-line drawing, even if it contains long edges. However, these drawings may require up to exponential width [51]. For simplicity we assume that the level lines are drawn equidistant. Let $e = (u, v) \in E$ be a (potentially long) edge which is replaced by a directed *edge path* $p(e) = (u =$

$v_1, v_2, \ldots, v_{\text{span}(e)}, v_{\text{span}(e)+1} = v)$ where $v_2, \ldots, v_{\text{span}(e)}$ are dummy vertices. Solving (1.10) delivers the desired result [96].

$$\begin{aligned} &\min \sum_{p(e) \text{ with } e \in E} \sum_{i=2}^{\text{span}(e)} \left(x(v_i) - \frac{i-1}{\text{span}(e)} \left(x(v_{\text{span}(e)+1}) - x(v_1) \right) \right)^2 \\ &\text{subject to} \\ &\quad x(z) - x(y) \geq \delta \quad \text{for} \quad \phi(y) = \phi(z), y \prec z \end{aligned} \tag{1.10}$$

Since the objective of (1.10) is a quadratic function, it can be solved for small instances only. For planar level graphs Eades et al. [50, 51] have shown that every not necessarily proper level planar graph has a planar straight-line drawing which can be computed in $\mathcal{O}(|V|^2)$ time.

Vertical Edges For nice drawings, there are two major objectives to consider [59]. First the drawings should be compact and second the edges should be as close to vertical as possible. Gansner et al. [73] model this problem as a linear program (1.11).

$$\begin{aligned} &\min \sum_{(u,v) \in E'} \Omega((u,v)) \cdot |x(u) - x(v)| \\ &\text{subject to} \\ &\quad x(z) - x(y) \geq \delta \quad \text{for} \quad \phi(y) = \phi(z), y \prec z \end{aligned} \tag{1.11}$$

$\Omega(e)$ denotes the priority to draw edge $e \in E'$ vertical. Therefore, the authors suggest higher priorities for inner segments: $\Omega(e) = 8$ for inner segments, $\Omega(e) = 2$ for outer segments incident to exactly one dummy vertex, and $\Omega(e) = 1$ for all other outer segments. The linear program can be interpreted as a rank assignment problem on a compaction graph $G_a = (V, \{(u,v) \colon u, v \in V_i, |\pi(v) - \pi(u)| = 1, 1 \leq i \leq k\})$ with length function δ. Each valid rank assignment corresponds to a valid drawing. The objective function can be modeled by adding vertices and edges to G_a [73, 96].

The drawback of the vertical edges approach is, that edges can have as many bends as dummy vertices. This creates sometimes a "spaghetti" effect which reduces the readability. To avoid this negative behavior the *linear segments model* was proposed, where each edge is drawn as polyline with at most three segments. The middle segment is always drawn vertical. In general, linear segment drawings have less bends but need more area than drawings in other models. There have been some algorithms proposed for this model [24, 28, 129]. The approach of Brandes and Köpf [24] produces pleasing results in linear time. It is our favorite algorithm.

Brandes/Köpf There are several practical algorithms for horizontal coordinate assignment [24, 28, 54, 56, 74, 130, 131, 138] using different approaches for the optimization of various objective functions or iterative improvement techniques. However, most interesting is the established algorithm of Brandes and Köpf [24], which generates at most two bends per edge and draws every inner segment vertically if no two inner segments cross. Further, it minimizes the horizontal stretch of segments and also gives good results for the other aesthetic criteria. The algorithm needs $\mathcal{O}(|V'| + |E'|)$ computation time for proper level graphs $G' = (V', E', \phi)$ and is fast in practice.

The input is the embedding generated in the previous phase three of a proper level graph. The algorithm consists of three steps, where its first two *block building* and *compaction* are carried out four times (*runs*) and the third *balancing* combines the results. We describe the left top run only. The other three runs are symmetric and can be realized by flipping the graph horizontally and/or vertically.

For the block building the objective is to vertically align each vertex with its left upper median neighbor. Note, the median of an even sized set is not unique. First, all segments are removed that do not lead to an upper median neighbor, see Fig. 1.2(b). Then two alignments are conflicting if their edge segments cross or share a vertex. *Type 2 conflicts*, two crossing inner segments, are assumed to have been avoided by the crossing reduction phase and not to occur, e. g., using the barycenter method from Sect. 1.2.3.1. *Type 1 conflicts*, an outer segment crossing an inner segment, are resolved in favor of the inner segment, i. e., the outer segment is removed from the graph. *Type 0 conflicts*, two crossing outer segments, are resolved greedily in a leftmost fashion, i. e., the right segment is removed from the graph. At this point there are no crossings left, see Fig. 1.2(c).

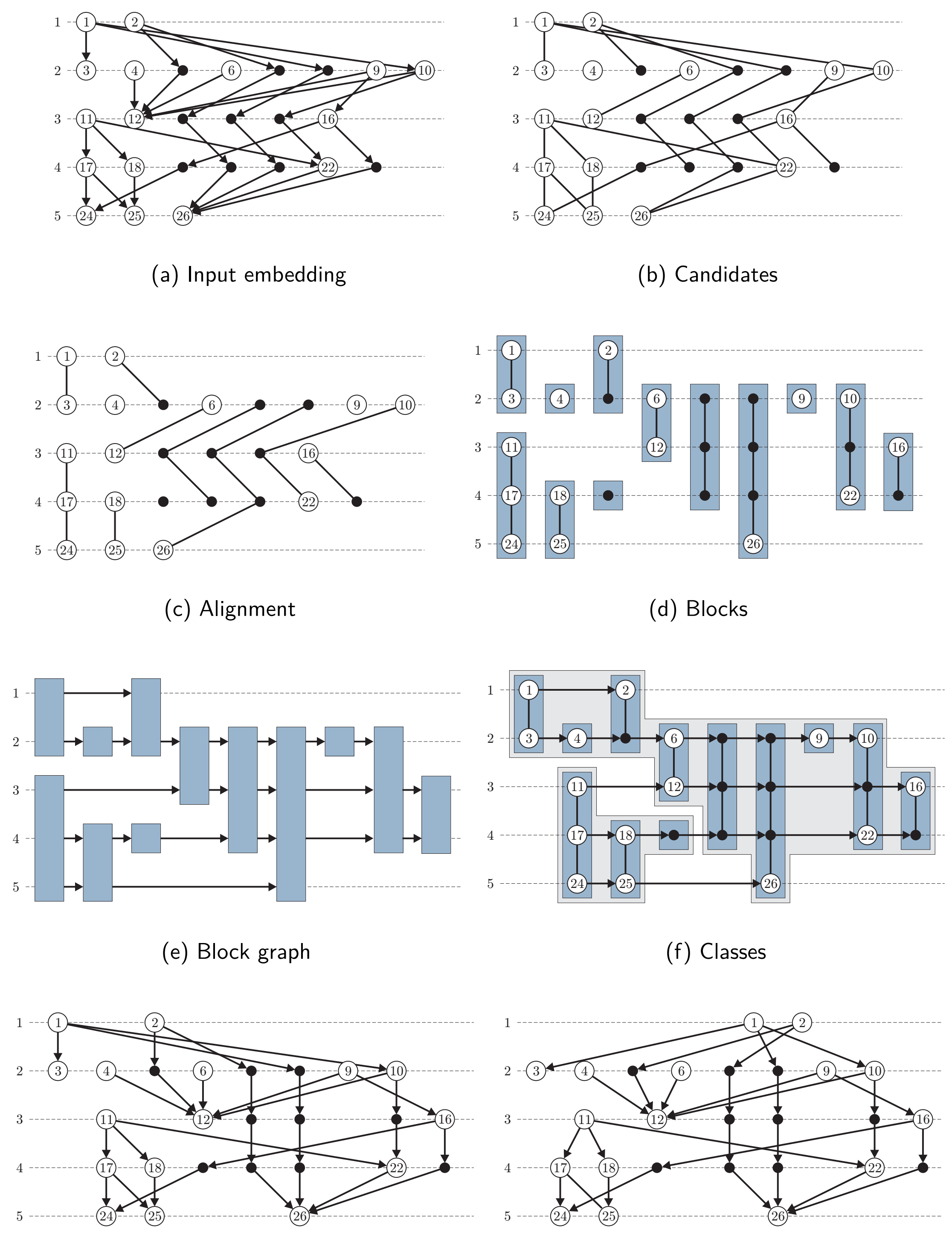
For the (horizontal) compaction each connected component is combined into a *block*, see Fig. 1.2(d). Consider the *block graph* obtained by introducing directed edges between each vertex and its successor (if any) on its level, see Fig. 1.2(e). We call this edges *inter-block edges*. A "horizontal" longest path leveling by topologically sorting the block graph determines the x-coordinate of each block and thus of each contained vertex. Thereby, the given minimum vertex separation δ is preserved. The block graph with expanded blocks is partitioned into *classes*, see Fig. 1.2(f). The first class is defined as the set of vertices which are reachable from the top left vertex. Then the class is virtually removed from the block graph. This is repeated, until every vertex is in a class. Within the classes the graph is already compact. Now the algorithm places the classes as close as possible. In Fig. 1.2(f) this already happened. Fig. 1.2(g) shows the complete left upper layout.

Finally each vertex has four x-coordinates. In the balancing step the two left (right) aligned assignments are shifted horizontally so that their minimum (maximum) coordinate agrees with the minimum (maximum) coordinate of the smallest width layout. The resulting coordinate is the average median[3] of the four intermediate coordinates. Fig. 1.2(h) shows the resulting drawing.

1.3 Miscellaneous

Counting the absolute number of crossings χ of edges between two levels in a given embedding of a two-level graph $G = (V, E, \phi)$ can be done by a naive sweep line

[3]The average median is defined as the average of the possible median values.



(a) Input embedding

(b) Candidates

(c) Alignment

(d) Blocks

(e) Block graph

(f) Classes

(g) Left upper layout

(h) Final balanced layout

Figure 1.2. Stages of the Brandes/Köpf algorithm

algorithm in $\mathcal{O}(|E| + \chi)$ time [129]. The authors of [15, 142] improved this time bound to $\mathcal{O}(|E| \log |V|)$.

An alternative approach for one-sided two-level crossing reduction is *planarization* [113, 114, 139], i. e., to remove a minimum number of edges such that the resulting graph is level planar. After drawing this embedding the removed edges are reinserted which, however, may produce many crossings. The planarisation problem is $\mathcal{NP}$-hard for both two-sided and one-sided cases, and can be solved in polynomial time for the case of two given fixed orderings [57]. For an integer programming solution for the two-layer planarization problem see [115]. The proposed advantage of planarization is that the drawings generated this way are more readable for humans [114]. However, this is not completely clear up to now.

For the *two-sided two-level crossing minimization*, i. e., both consecutive levels are permutable, Shahrokhi et al. [133] proved that the problem is $\mathcal{O}(\log |V|)$-approximable, when the maximum degree over the minimum degree is bounded by a constant. Jünger and Mutzel [92] presented an ILP for two-sided crossing minimization problem. Recently, Buchheim and Zheng presented quadratic programming methods for this problem [29].

1.4 Overview

After some preliminaries and an extensive introduction to the hierarchical framework, we discuss some new insights to the generation of hierarchic drawings in the next Chapter 2. This includes how to deal with edges between vertices within the same level and a practical global approach for k-level crossing reduction. Chapters 3 and 4 summarize our adaptions of the specific phases of the Sugiyama framework for drawing on concentric and radial level lines. For an easy understanding, these chapters describe our investigations and new results high level. The detailed algorithms, proofs, and analyses can be found in the articles listed in Appendix A. We conclude in Chapter 5 with a short recapitulation of the major results and provide some suggestion for future investigations and research in the wider context of level graphs.

2 Horizontal Drawings

In this chapter we extend the hierarchical framework with classic horizontal level lines. First we show a new extension how to consider edges between vertices within the same level [11, 13] and then show a new global crossing reduction based on sifting [4]. For the latter, we consider all levels simultaneously and are not afflicted by the disadvantages of the algorithms listed in Sect. 1.2.3.2. For an easy distinction with the contents of the following chapters, we will from now on call level drawings with horizontal level lines *horizontal drawings* and prepend the phases with the phrase "horizontal" if necessary.

2.1 Crossing Reduction

For our horizontal drawings we only extend the crossing reduction phase described in Sect. 1.2.3 for some new features. The other phases essentially remain unchanged.

2.1.1 Intra-Level Edges

Many real world graphs exhibit hierarchies with edges between the vertices on the same level. For example, the visualization of centrality of actors in social networks [26] produces level graphs with both (directed) *inter-level* edges with a span of at least one and *intra-level* edges with a span of zero, i. e., (undirected) edges connecting vertices within the same level. See Fig. 2.1 for an example. Drawing these *extended level graphs* was addressed as one of the open problems in social network visualization by Brandes [20]. Another example are UML class diagrams whose levels are traditionally determined by the depth of inheritance. There may be somewhat arbitrary associations or uses-relations between the classes, especially between classes on the same level. It has neither been shown yet that intra-level edges reduce the visual complexity nor that they reduce the total number of crossings. However, the fact that all existing hierarchical drawing methods more or less suppress intra-level edges, although they are present from the application in many cases, justifies an investigation.

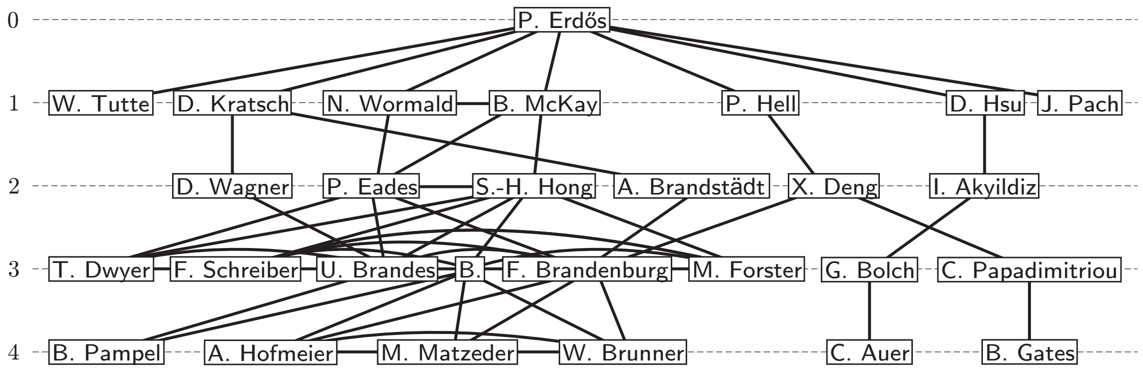


Figure 2.1. Extract from the coauthor graph of the famous Hungarian mathematician Paul Erdős. The levels correspond to *link distances* as BFS levels starting at Erdős, inter-level edges are BFS tree edges, and intra-level edges are BFS traverse edges

It is not surprising that the *extended one-sided two-level crossing minimization problem* remains $\mathcal{NP}$-hard. Hence, we show how to enhance the promising one-sided two-level sifting heuristic from Sect. 1.2.3.1 to include intra-level edges and to consider all types of the arising crossings for reduction, i. e., crossings of two inter-level edges, an inter-level and an intra-level edge, and between two intra-level edges. See Fig. 2.2 for an example. We call our algorithm *extended sifting*.

In our *extended level drawings* of extended k-level graphs $G = (V, E, H, \phi)$, we represent intra-level edges in $H = H_1 \dot\cup H_2 \dot\cup \ldots \dot\cup H_k$ using circular arcs with different radii in order to avoid overlapping edges and crossings between vertices and edges. Thereby, the respective amplitude of the only interpolation point in the middle raises with the number of enclosed vertices such that the gradients of the tangents in the endpoints are lower as the most shallow inter-level edge. This avoids unnecessary (double) crossings among intra-level and inter-level edges. Further, we restrict to draw the arcs only on one-side of the level lines, say above, in order to model the problem such that the Sugiyama framework can cope with it.

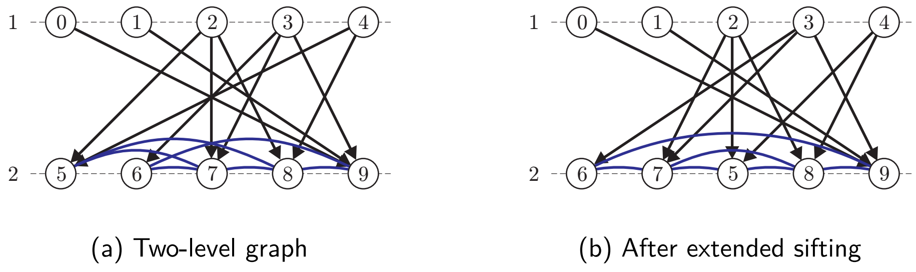


Figure 2.2. Extended one-sided two-level crossing reduction

2.1.1.1 Circular Sifting

We do not use the traditional sifting method operating on the crossing matrix but apply the ideas of Baur and Brandes [16] used for the $\mathcal{NP}$-hard [107] crossing minimization problem in *circular drawings*. This is finding an ordering for the vertices V of a graph $G = (V, E)$ which all are placed on a single circle to obtain a reduced/minimum number of crossings among the straight-line edges in E. Since there does not exist a "circular" order in mathematics, Baur and Brandes define linear orders $\prec_\alpha$ by selecting a reference vertex $\alpha \in V$ as the first of the (here w. l. o. g. counter-clockwise) sequence. For finding the locally optimal position of a vertex $u \in V$ in a sifting step, it is sufficient to record the change in crossing count Δ while swapping u with its successor $v_p \in V$. This can be done by considering only edges incident to u or v_p: After a swap exactly those pairs of these edges cross which did not cross before. All other crossings remain unchanged. This is shown by (2.1) more precisely, where $\prec'_\alpha$ is the resulting ordering/*circular embedding* after the swap.

$$\Delta = \sum_{t \in N(v_p)} \Big| \{ s \in N(u) \mid s \prec'_t v_p \} \Big| - \sum_{s \in N(u)} \Big| \{ t \in N(v_p) \mid t \prec_s u \} \Big| \tag{2.1}$$

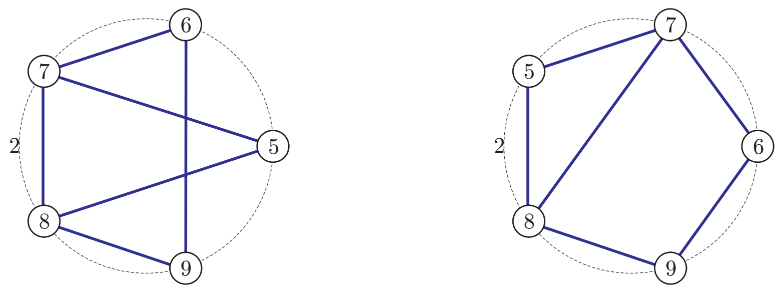
At the end of one step, u is placed where the intermediary crossing counts reached their minimum. For efficiency reasons, the computation of Δ is implemented over suffix lengths in ordered adjacency lists. For this, the adjacency list of each vertex v must be kept sorted according to the current order $\prec_v$ of the positions of v's neighbors. Circular sifting needs $\mathcal{O}(|V| \cdot |E|)$ time for one round.

2.1.1.2 Extended Sifting

The same holds for swapping two consecutive vertices u and $v_p \in V_i$ with $1 < i \leq k$ in horizontal one-sided two-level sifting of a graph $G^* = (V_{i-1} \dot{\cup} V_i, E_i, H_i, \phi) \subseteq G = (V, E, H, \phi)$, which is proper, considering only inter-level edges E_i: After a swap exactly those pairs of edges of $\{(\cdot, u)\} \subseteq E_i$ and $\{(\cdot, v_p)\} \subseteq E_i$ cross which did not cross before and all other crossings remain unchanged. Thus, similar to [16] we can efficiently compute the difference in the number of crossings Δ_E before and after the swap in $\mathcal{O}(|N^-(u)| + |N^-(v_p)|)$ time by keeping the adjacency lists of the vertices in V_i ordered according to the ascending positions of their neighbors in V_{i-1}. In a sifting step we only have to record the position which led to a lowest value of Δ_E and then have found the locally optimal position to place the traversing vertex u. Since we need not necessarily know the absolute numbers of crossings per swap, we can omit the $\mathcal{O}(|E_i|^2)$ time consuming creation and the further maintenance of the crossing matrix. Actually, we again use (1.3) for recording the number of crossings within a sifting step, but set the initial value of $\chi_{\mathcal{E}} = 0$. With the described method we exactly obtain the same results, i. e., the same orderings of V_i, in $\mathcal{O}(|V_i| \cdot |E_i|)$ time. From now on we call this new type of sifting *relative sifting* to stress the differences to the traditional *absolute sifting* using the crossing matrix.

Hence, it remains to compute the differences in the two remaining types of edge crossings, i. e., between two intra-level edges in H and between inter-level edges in E and intra-level edges in H. Although there is also a compact method [11], we compute the respective differences independently and add them afterwards for a total difference $\Delta = \Delta_E + \Delta_H + \Delta_{EH}$ per swap. This modular system gives us more transparency and flexibility to use the same approach for the other types of hierarchic drawings described in the following chapters.

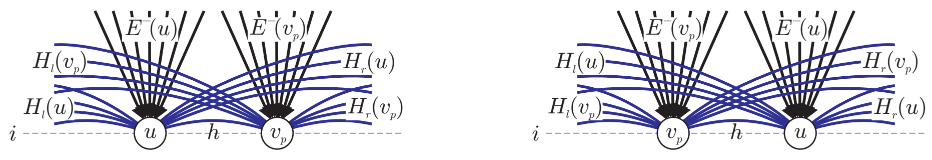
For the treatment of crossings between intra-level edges, we directly use circular sifting: Consider the line of level i logically bent to a circle, e. g., as it is done in Fig. 2.3(a) for the intra-level edges of the graph in Fig. 2.2(a). Since circular sifting is also naturally modularized by elementary sifting steps, its Δ_H computation of crossings directly delivers the difference in crossings of intra-level edges when swapping two vertices on the horizontal level i.



(a) Before (b) After

Figure 2.3. Circular crossing reduction for the intra-level edges of the graph in Fig. 2.2

Now we treat crossings between inter-level and intra-level edges. Again, swapping u with its current successor v_p changes only crossings among pairs of inter-level and intra-level edges which are incident to u or v_p. Thus, for computing the change in the crossing count, we only need the sizes of the six sets $H_l(v)$, $H_r(v)$, $E^-(v)$ with $v \in \{u, v_p\}$, see Fig. 2.4. $H_l(v) = \{\,\{u,v\} \mid u \prec v\,\}$ is the left adjacency and $H_r(v) = \{\,\{v,w\} \mid v \prec w\,\}$ is the right adjacency for each vertex $v \in V_i$. Each of them is sorted according to the current ordering of V_i with the traversing vertex u in the front.



(a) Before swapping (b) After swapping

Figure 2.4. Crossings among inter-level and intra-level edges

Neglecting potentially existing short edges $h = \{u, v_p\} \in H$ which is a non-contributing special case, we obtain (2.2) as change in crossing count Δ_{EH} when swapping u and its direct successor v_p.

$$\Delta_{EH} = (|H_r(v_p)| - |H_l(v_p)|) \cdot |E^-(u)| + (|H_l(u)| - |H_r(u)|) \cdot |E^-(v_p)| \qquad (2.2)$$

During a sifting step we have to keep the ordering of the adjacency of the traversing vertex u up to date while swapping with successors v_p. This is easy, because an affected edge $h = \{u, v_p\}$ which must be added as the last element to $H_l(u)$ is always the first of $H_l(v_p)$. Note, the orderings of the intra-level adjacency lists of all vertices v_p are valid throughout the complete sifting step besides obsolete positions of edges $\{u, v_p\}$. However, these exceptions are irrelevant for the correctness.

We obtain an overall time complexity of $\mathcal{O}((|V_{i-1}| + |V_i|) \cdot (|E_i| + |H_i|))$ for one sifting round [11]. In a top-down sweep for a proper k-level graph $G = (V, E, H\phi)$, we reorder the levels i from 2 to k by consecutively applying our extended one-sided two-level crossing minimization on the fix ordered set V_{i-1} and on the freely permutable set V_i. In the subsequent bottom-up sweep we reorder the levels i from $k-1$ down to 1 by consecutively applying it on the fix ordered set V_{i+1} and the permutable set V_i. However, in the bottom-up sweep we have a slightly different situation, since the intra-level edges are above the current level i and cross edges from level i and $i-1$. Nevertheless, the formula for crossings of intra-level and inter-level edges (2.2) does not depend on any vertex ordering different to that on level i and especially does not depend on that of level $i-1$. Thus, we count the change in the number of crossings of the intra-level edges of level i with the inter-level edges between level i and $i-1$ during a swap. For this, we take for each vertex $v \in V_i$ the set $E^+(v)$ instead of $E^-(v)$ in (2.2). After some iterations, say 10, of top-down with subsequent bottom-up sweeps the algorithm terminates in $\mathcal{O}(|V| \cdot (|E| + |H|))$ time.

2.1.2 Global Sifting

For nearly thirty years, the common crossing reduction technique has been to consider only two consecutive levels at a time and to iterate them in multiple top-down and bottom-up sweeps. Although this procedure has received much attention, there is a need for a new technique, because the level-by-level approach may be stuck in a local optimum. This is even the case if the one-sided two-level crossing reduction method in fact is a minimization and returns optimal vertex orderings of the levels in the sense of a minimum number of crossings, e. g., see Fig. 2.5. This may even hold for infinitely many top-down with subsequent bottom-up sweeps, since the example returns the same embedding after the second bottom-up sweep as obtained from the first bottom-up sweep and, thus, runs in a circle potentially without any improvement. The idea to find such an example is the following: It may be better to have slightly more crossings between levels 1 and 2 and 3 and 4 than the minimum, but then potentially have noticeable fewer crossings in the middle between levels 2 and 3. Bastert and Matuszewski claim in [96, page 102] that the results are even far from optimum. Alas, potentially existing approximation ratios of one-sided two-level reduction algorithms do not translate to k-level graphs: Apart from the extremes, levels are not fixed from the input. Their vertex orders are results of previous applications of the two-level algorithm.

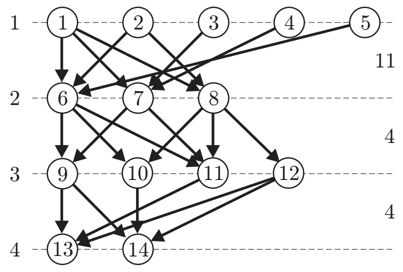
An important feature of crossing reduction algorithms is the avoidance of type 2 conflicts. Among others, the standard fourth phase algorithm by Brandes and Köpf (Sect. 1.2.4) assumes the absence of type 2 conflicts to align long edges vertically. Since this is one of the major aesthetic criteria for easily readable hierarchical drawings [96] and is dependent of the crossing reduction results, we support this in our global sifting algorithm inspired by [24, 59]. The simple level-by-level barycenter and median heuristics from Sect. 1.2.3.1 avoid type 2 conflicts automatically in horizontal layouts. Although two-level algorithms reduce the crossings between V_{i-1} and V_i, the number of crossings between V_i and V_{i+1} (and thus even the total number of crossings) can increase while permuting V_i. These heuristics push the crossings downwards or upwards until they are resolved at level k or 1, respectively. This is not possible with cyclic levels anymore and, thus, many crossings will remain after the level where the cyclic sweep was stopped. Even worse, there may remain type 2 conflicts. Alternatives are ordered k-level sifting or the similar centered three-level crossing reduction described in Sect. 1.2.3.2. However, both generate many type 2 conflicts and are for reaching a global optimum both restricted to a local view. Thus, they also may tend to get stuck in local optima.

Our idea has some similarities to the *sparse normalization* of Eiglsperger et al. [59] which also guarantees the absence of type 2 crossings. However, we go one step further: We do not use the sweeps with two-level one side fixed crossing reduction. We consider the *block graph* – which is the same as their *compaction graph* completed with one sweep over all levels – directly for crossing reduction. That is, we treat the graph as a whole. This is new for crossing reduction. Eiglsperger et al. obtain an overall running time in $\mathcal{O}((|V| + |E|) \log |E|)$ for a non-necessarily proper k-level graph $G = (V, E, \phi)$ using the *splay tree* data structure of Sleater and Tarjan [135] for their operations. Unfortunately, their algorithm has the limitation that it can only use crossing reductions which solely consider the position numbers of the fixed neighbors as input for the position computations of the vertices in the permutable set. As a consequence, they only can use simple heuristics like barycenter or median instead of the more advanced sifting heuristic for example. They obtain identical results as the slower sweeping barycenter and median approaches whose running times depend on the number of dummy vertices. In a nutshell, Eiglsperger et al. optimize speed but not quality. In our approach we use the blocks to improve the quality of the resulting embedding.

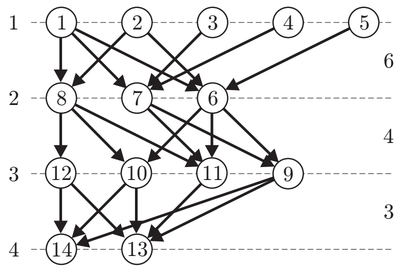
2.1.2.1 Algorithm

We treat each vertex and each maximum connected subgraph consisting only of dummy vertices as a *block*[1], e. g., see Fig. 2.6(a), and execute sifting on all these blocks at the same time independent of their levels. The algorithm gives better results than

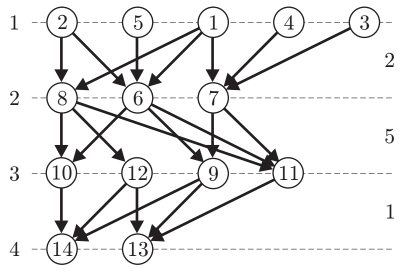
[1]The blocks for assigning coordinates in Sect. 1.2.4 are defined slightly different, e. g., they may contain both dummy and original vertices.



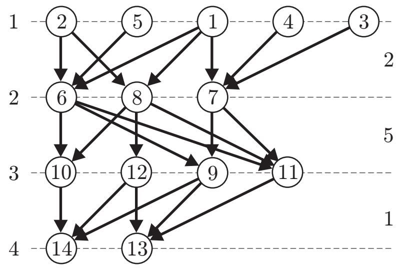
(a) Input embedding, $\chi = 19$



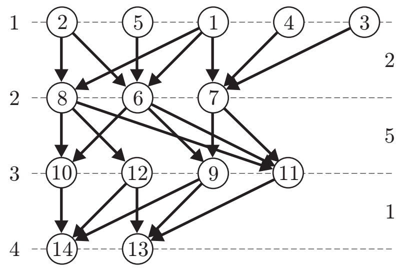
(b) After first top-down sweep, $\chi = 13$



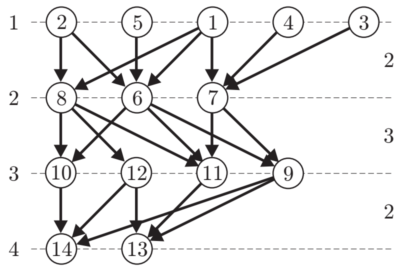
(c) After first bottom-up sweep, $\chi = 8$



(d) After second top-down sweep, $\chi = 8$



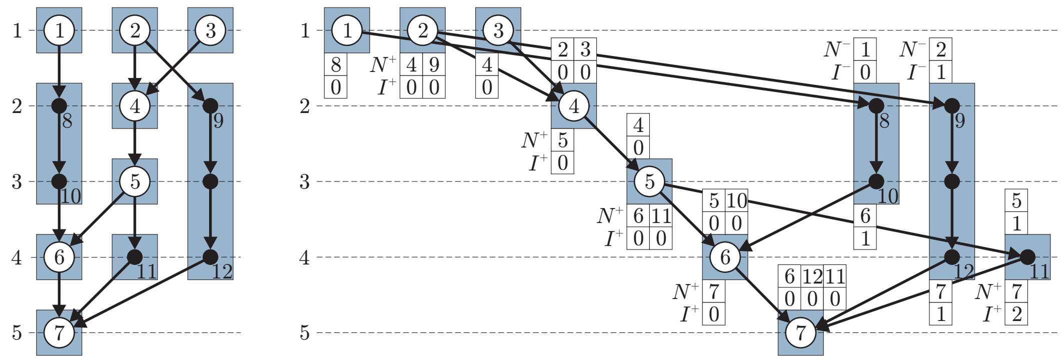
(e) After second bottom-up sweep, equals (c)



(f) Optimal embedding, $\chi = \chi_{OPT} = 7$

Figure 2.5. Crossing reduction using the level-by-level sweep method has been stuck in local minimum

traditional sifting heuristics, is easily applicable to other problems, and has quadratic time complexity independent of the number of dummy vertices.



(a) A level graph with ten blocks

(b) Separate π-positions, ordered adjacency lists N^- and N^+, and index arrays I^- and I^+ for each block

Figure 2.6. Blocks as sifting objects

Let $\mathcal{B}$ be the list of all blocks of the graph and the bijective function $\pi\colon \mathcal{B} \to \{0, \dots, |\mathcal{B}|-1\}$ define the current *positions* of the blocks in $\mathcal{B}$. As an initialization, we use an arbitrary ordering. Now we perform relative sifting on the *block graph*[2] which consists of the blocks as vertices and their incident outer segments (which span exactly one level) as edges. Thereby, the positions of the blocks are defined by π. We perform a conventional sifting step for a block under the assumption that the positions of all other blocks are fixed, i. e., we test all positions for a *traversing block* and move it to the best possible position in which there were the fewest crossings. Since we treat all dummy vertices of each edge (and each original vertex) as one block and try to find the best position for the entire block in one step, we eliminate the problems of traditional two-level crossing approaches which lack this global view of the crossings of a long edge.

Again, we use relative sifting as described at the beginning of Sect. 2.1.1.2. What remains is to show how the change in the number of crossings Δ is computed when swapping two consecutive blocks. For each swap we have to consider incident edges between two pairs of levels. One level pair of interest is exactly above the block starting at the higher level, whereas the other level pair is exactly below the block stopping at the lower level. Between all other pairs of levels there are no changes. We treat both level pairs separately and add the computed change in crossings afterwards. Note that outer segments may cross inner segments even if they did not before, i. e., blocks can be crossed in their middle part. These type 1 conflicts are tolerated, however, must be counted for Δ. As we use for the concrete computation the respective vertices of the involved blocks instead of the entire blocks, we can directly apply the algorithm of [16] as we have done it in one-sided two-level sifting. For this, we have to keep the adjacencies

[2]The block graph in Sect. 1.2.4 for coordinate assignment is different, e. g., contains other blocks and directed edges between successors on the same level.

ordered according to ascending positions π, as indicated by the $N^{\cdot}$ arrays in Fig. 2.6(b). For a block $A \in \mathcal{B}$ let $\deg(A)$ be the number of incident outer segments to any vertex in A. Since we maintain additional *index arrays* $I^{\cdot}$ indicating where each block is stored in the adjacencies of its neighbors, see Fig. 2.6(b), we efficiently can perform one sifting swap of two consecutive blocks $A, B \in \mathcal{B}$ in $\mathcal{O}(\deg(A) + \deg(B))$ time.

One may claim, that there may be a large overhead for useless sifting swaps of blocks whose positions are *independent* of each other, i. e., which are located on disjoint levels and have no common neighbors. However, since a swap of two independent blocks needs only constant time, it would at least consume the same time to check, that a swap might be unnecessary. Thus, we omit that check to keep the algorithm fast and simple.

Finally, for the resulting embedding we set each vertex to the position of its block and order the levels according to this positions. We obtain an overall running time in $\mathcal{O}(|E|^2)$ for one sifting round of a not necessarily proper level graph $G = (V, E, \phi)$. This parallels the linear running time $\mathcal{O}(|V|)$ of the level planarity test [91] which also is independent of (the number of) dummy vertices. In practice, ten sifting rounds suffice.

2.1.2.2 Simple Global Heuristics

Using the idea of blocks, we also extend the simple barycenter and median crossing reduction strategies: We iteratively take the given π-positions of the blocks in $\mathcal{B}$ and for each block compute the barycenter or median, respectively, and sort $\mathcal{B}$ according to these values. Notice, if phases two or four of the Sugiyama framework need dummy vertices, then a pre- and postprocessing in $\mathcal{O}(|V'|) \subseteq \mathcal{O}(|V| + k \cdot |E|)$ time is needed where $G' = (V', E', \phi')$ is the normalized version of a k-level graph $G = (V, E, \phi)$. Our benchmarks in Sect. 2.1.2.3 show that both are very fast but qualitatively not competitive with global sifting in the number of crossings.

2.1.2.3 Benchmarks

Now we experimentally compare the practical performances of the different algorithms using the Java implementations of [8, 81]. In a nutshell, the simple global approaches are fast but deliver no acceptable results. Traditional level-by-level sifting is fast, leaves few type 2 conflicts, but many crossings, however. Centered three-level sifting and ordered k-level sifting are fast, leave few crossings, but many type 2 conflicts. Global sifting produces even less crossings than both without any type 2 conflicts at the cost of a slightly higher running time, which is still feasible. The running time of global sifting is independent of the number of dummy vertices. This parallels the advanced algorithm of [59]. With a dummy vertex proportion of 75% the running time is about twice as high as needed for traditional sifting.

Figures 2.7 to 2.11 show the practical performance of the algorithms iterative one-sided two-level barycenter (B), median (M), and sifting (S), iterative centered three-level sifting (3S), ordered k-level sifting (OS), and global barycenter (GB), global median (GM), and global sifting (GS) applied to the normalized versions $G' = (V', E', \phi')$ of graphs $G = (V, E, \phi)$. For comparability reasons, we have never used the lower bound speed-up of [77]. For each of the former four heuristics we applied ten top-down with subsequent bottom-up sweeps and for each of the latter ones we performed ten rounds. For each graph size, we have generated ten arbitrary graphs G' with an aspect ratio, i. e., maximum number of vertices per level vs. number of levels, of the *golden rectangle*, i. e., of $\frac{1+\sqrt{5}}{2}$. We omit measurements of the fast algorithm of Eiglsperger et al. [59], because the obtained embeddings and crossings are the same as left by sweeping barycenter or median, respectively. All benchmarks were run on a 2.83 GHz Intel XEON workstation under Solaris 10 with the Java 6.0 platform of Sun Microsystems, Inc.

We compare the different running times in Fig. 2.7. Although global sifting is the slowest, it is still feasible in practice even for interactive applications. Clearly, its performance on graphs from practice is not as dire as the worst case $\mathcal{O}(|E|^2)$ time complexity may suggest on a first glance.

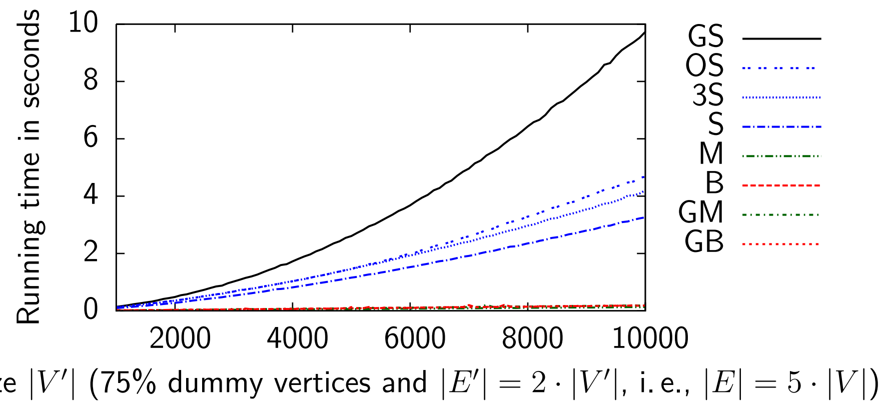


Figure 2.7. Running times

Figure 2.8 shows the quality of the heuristics in the number of crossings of the resulting embeddings. The results of global sifting are about five to ten percent better than the ones of the established algorithms. This does not seem to be very much with respect to the higher running times, but remember, contrary to the traditional sifting algorithms the resulting embeddings do not contain any type 2 conflicts. See Fig. 2.10 for that.

Figure 2.9 depicts that the traditional approaches have a constant running time on graphs with the same sizes but ascending proportions of dummy vertices. For global sifting the running times get the better the more vertices are dummies, i. e., the more inner segments exist which are treated as whole. This reflects that the time complexity $\mathcal{O}(|E|^2)$ depends only on the number of long edges $|E|$ rather than on the number of segments $|E'|$ of the normalized version of the graph.

In Fig. 2.11 we try to compare the results of the heuristics with the exact solution. For an in practice solvable ILP of the exact algorithm as described in Sect. 1.2.3.2, we had to keep the graphs with $|V'| \leq 35$ small. Since we have a reasonable proportion of 30% dummy vertices, the graphs are rather thin. This may be the reason why barycenter here outperforms two sifting algorithms. Figure 2.8 indicates that under normal circumstances sifting is qualitatively the better choice [96].

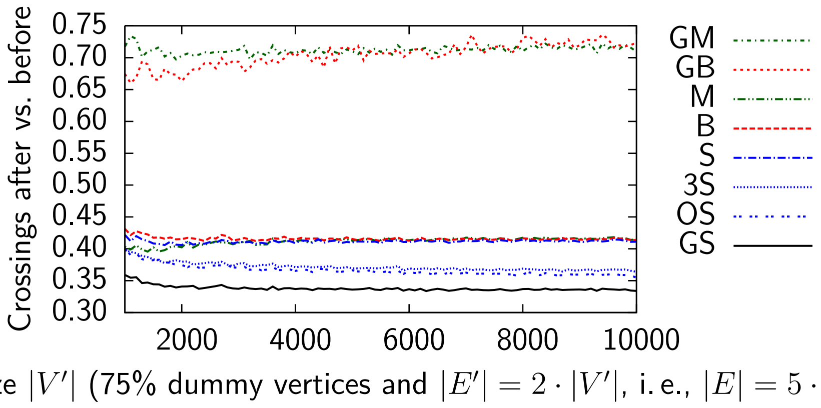


Figure 2.8. Number of crossings after vs. before applying the crossing reduction

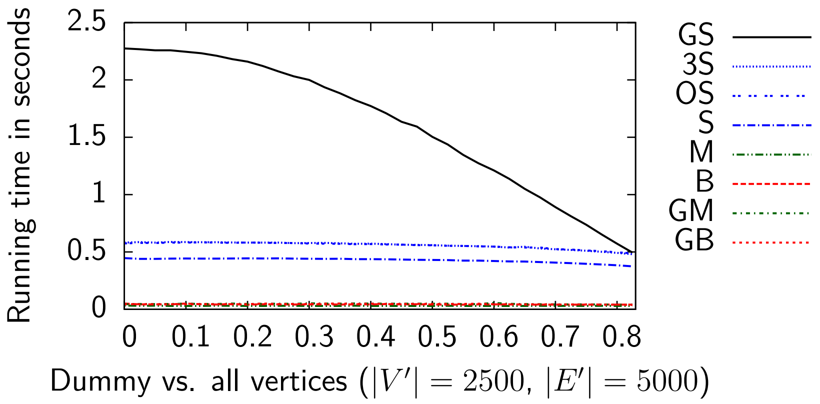


Figure 2.9. Running times with different proportions of dummy vertices

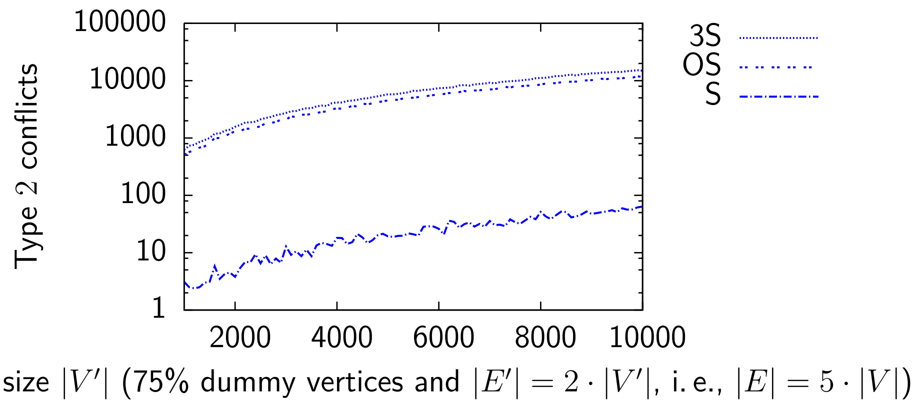
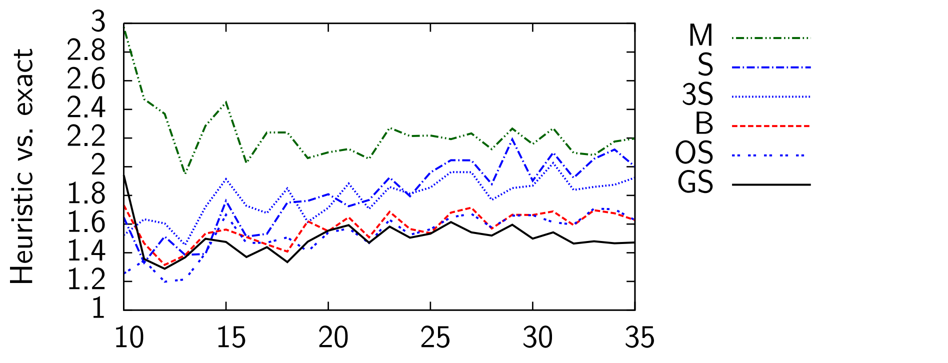


Figure 2.10. Number of type 2 conflicts



Graph size $|V'|$ (30% dummy vertices and $|E'| = 1.25 \cdot |V'|$, i.e., $|E| \approx 1.36 \cdot |V|$)

Figure 2.11. Number of crossings compared to optimal solutions (by ILP)

2.1.3 Exact Crossing Minimization without Type 2 Conflicts

Type 2 conflicts can also be excluded by the exact crossing minimization of Jünger et al. [90] described in Sect. 1.2.3.2 by adding the additional constraints of (2.3). Here, let $E' \subset E$ be the set of all inner segments. Remember, a value $c^i_{uvwy} = 0$ means that (inner) edges $(u, v), (w, y) \in E'$ do not cross.

$$c^i_{uvwy} = 0 \quad \text{for} \quad \begin{array}{l} (u,v),(w,y) \in E_i, \\ 1 \leq u < w \leq |V_{i-1}|, \\ 1 \leq v < y \leq |V_i|, \\ (u,v),(w,y) \in E', \\ 1 < i \leq k \end{array} \tag{2.3}$$

This results after simplification in a similar ILP to our approach which will be described next. However, using our blocks gives a more direct correspondence to the ILP, provides deeper insights, and avoids the usage of dummy vertices for computing a solution of the ILP[3]. For this reason and for a typically lower number of blocks compared to the number of vertices, the formulation needs fewer variables which potentially leads to lower computation times.

For the direct approach, we start with an arbitrary but fixed ordering of the list of blocks $\mathcal{B}$. For any two blocks A, B with $\pi(A) < \pi(B)$ which have a common level we define a boolean variable x_{AB}. The value of $x_{AB} = 1$ denotes that A is left of B and $x_{AB} = 0$ that B is left of A in the final embedding. For each triple of blocks A, B, C with $\pi(A) < \pi(B) < \pi(C)$ with at least one common level, we add the condition $0 \leq x_{AB} + x_{BC} - x_{AC} \leq 1$ to exclude cyclic dependencies within a level.

Let $s_1 = (a, b)$ and $s_2 = (c, d)$ be segments between the same levels such that at least one of them is an outer segment. Let $A = \text{block}(a), B = \text{block}(b), C = \text{block}(c), D = \text{block}(d)$ be the blocks which contain the vertices a, b, c, d. Note that $A = B$ or $C = D$ holds if s_1 or s_2 is an inner segment, respectively. W. l. o. g. let $\pi(A) < \pi(C)$. We add a boolean crossing variable c_{ABCD} which indicates whether s_1

[3]For the determination of the equations the graph must be proper, however.

and s_2 cross. If $\pi(B) < \pi(D)$ we add the constraint $-c_{ABCD} \leq x_{BD} - x_{AC} \leq c_{ABCD}$ otherwise we add $1 - c_{ABCD} \leq x_{DB} - x_{AC} \leq 1 + c_{ABCD}$. The objective function is to minimize the sum of the values of all crossing variables. Let $\text{levels}(A) \subseteq \{1, \ldots, k\}$ be the set of levels which contain a vertex of a block $A \in \mathcal{B}$ or which are spanned by its long edge if omitting dummy vertices. See (2.4) for the complete ILP formulation for a proper k-level graph $G = (V, E, \phi)$ with $E' \subset E$ denoting the set of inner segments. Informally speaking, each member of the set $\mathcal{C}$ denotes the up to four incident blocks of each pair of edges which potentially cross.

$$
\begin{aligned}
\chi_{\text{opt}} &= \min \sum_{(A,B,C,D) \in \mathcal{C}} c_{ABCD} \\
\mathcal{C} &= \{\, (A,B,C,D) \in \mathcal{B}^4 \mid \; \exists (a,b),(c,d) \in E: \\
&\qquad (a,b) \notin E' \vee (c,d) \notin E', \\
&\qquad \phi(b) = \phi(d), \\
&\qquad A = \text{block}(a), B = \text{block}(b), \\
&\qquad C = \text{block}(c), D = \text{block}(d), \\
&\qquad \pi(A) < \pi(C) \,\}
\end{aligned}
$$

subject to

$$
\begin{array}{lll}
-c_{ABCD} \leq x_{BD} - x_{AC} \leq c_{ABCD} & \text{for} & (A,B,C,D) \in \mathcal{C}, \\
 & & \pi(B) < \pi(D) \\
1 - c_{ABCD} \leq x_{DB} - x_{AC} & & \\
\quad \leq 1 + c_{ABCD} & \text{for} & (A,B,C,D) \in \mathcal{C}, \\
 & & \pi(B) > \pi(D) \\
0 \leq x_{AB} + x_{BC} - x_{AC} \leq 1 & \text{for} & A,B,C \in \mathcal{B}, \\
 & & \pi(A) < \pi(B) < \pi(C), \\
 & & \text{levels}(A) \cap \text{levels}(B) \\
 & & \quad \cap \text{levels}(C) \neq \emptyset \\
x_{AB} \in \{0,1\} & \text{for} & A,B \in \mathcal{B}, \\
 & & \pi(A) < \pi(B) \\
c_{ABCD} \in \{0,1\} & \text{for} & (A,B,C,D) \in \mathcal{C}
\end{array}
\tag{2.4}
$$

2.1.4 Other Applications

Healy et al. [80, 82, 83] use a data structure called *vertex exchange graph* as a simple test for level planarity in $\mathcal{O}(|V'|^2)$ time for a normalized k-level graph $G' = (V', E', \phi)$ of $G = (V, E, \phi)$. Using our blocks it is straightforward to improve the running time of the test from $\mathcal{O}(|V'|^2)$, i. e., $\mathcal{O}((k \cdot |V|)^2)$ worst case, to $\mathcal{O}(|V|^2)$ similarly to the ILP in Sect. 2.1.3: Each pair of level overlapping blocks builds one vertex in the vertex exchange graph. Similar techniques can be used to reduce the number of 2-SAT clauses in [124] to test level planarity or to minimize crossings.

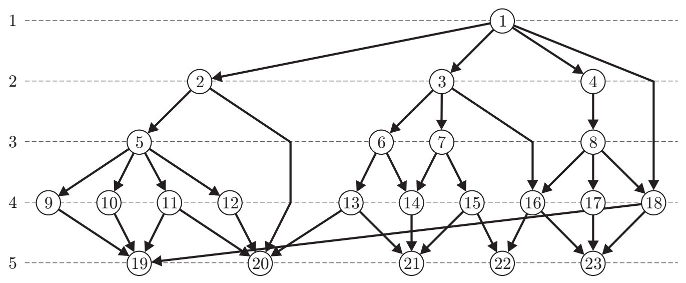
In a clustered level graph vertices are combined to subgraphs in a hierarchical way [68]. The crossing reduction has to ensure that all (dummy) vertices of a subgraph on the same level are consecutive and that all subgraphs spanning several levels have a matching ordering on each level to avoid crossings of subgraphs. This is rather complicated using a two-level crossing reduction approach [131]. Using global sifting this becomes simple: Instead of swapping a vertex with its right neighbor in a sifting swap we swap all blocks of a subgraph one by one with their successive blocks and determine the change in the number of crossings. Valid positions are only where no distinguished subgraphs overlap. The time complexity stays the same as for standard global sifting. If the layout of the subgraphs themselves is not fixed, then global sifting can be recursively applied to the subgraphs as well, e. g., performing a sifting round for each hierarchical layer. For good results this may be iterated top-down and bottom-up the cluster hierarchy.

3 Radial Drawings

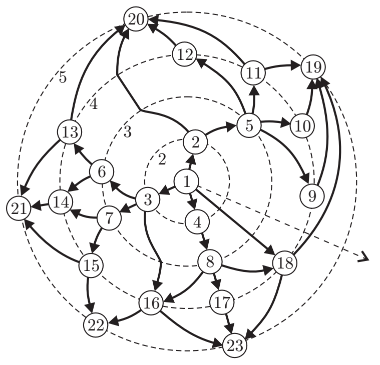
In *radial drawings* of hierarchical graphs we place the vertices on concentric circles instead of on horizontal lines and draw the edges as outwards monotone segments of spirals rather than straight lines as it is both done in the standard Sugiyama framework. Each curve lies between the concentric levels of its endpoints. This ensures that interior level lines are not crossed as it would be the case with straight lines. See Fig. 3.1 for a radial drawing (b) of the horizontal example in (a). The 3D drawing on a cylinder in Fig. 3.1(c) is topologically equivalent to (b). Logically, both latter illustrations indicate that there is no explicit left or right of a vertex on a radial level.

The radial drawing style is well suited for the visualization of social or policy networks, where structural centrality scores are mapped to geometric centrality or the display of the vicinity of a distinguished vertex is the primary intention [21–23, 25, 144]. See Fig. 3.2 for an example. Radial drawings allow a more flexible edge routing than horizontal drawings, as edges can be routed around the center in two directions. In our experimental results this reduces the number of crossings by approximately 30% on average compared to traditional horizontal drawings. It is also more likely that a graph can be drawn without any crossings at all, since the set of level planar graphs is a proper subset of the set of radial level planar graphs [1, 6, 7]. Radial level drawings are also well suited for level graphs with an increasing number of vertices on higher levels. For example, in a graph that shows which Web pages are reachable from a given start page by following k hyperlinks, higher levels are likely to contain many vertices while there are only few vertices on the lower levels. Other potential example applications can be found in [89, 148]. In the remainder of this chapter we describe how to obtain such drawing using the hierarchic framework.

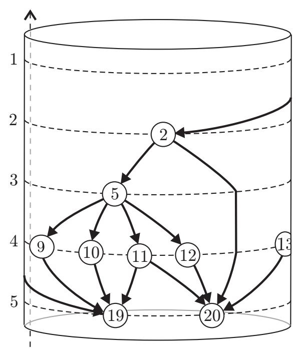
General radial drawings where vertices are constraint to concentric rings are not new, i. e., they date back to the 1940s and are called *ring* or *target diagrams* [25]. For some newer examples see the hierarchical graph drawings of [31], the ring diagrams of



(a) Horizontal



(b) Radial 2D



(c) Radial 3D

Figure 3.1. Drawings of a level graph

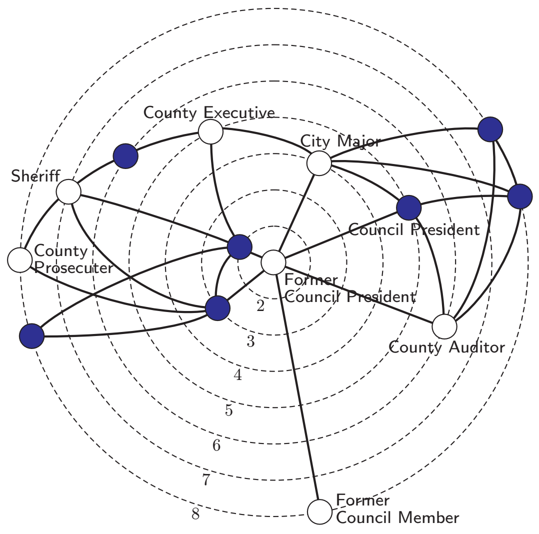


Figure 3.2. Political ties between prominent politicians of a county; the two apparent groups predict the voting pattern of City Council members (shaded vertices) on building a new prison (data from [21, 48])

[125], or the radial tree drawings of [49], which are common for free trees. Di Giacomo et al. [45, 46] study drawings of planar radial level graphs where the edges are polylines (with straight-line instead of monotone spiral segments). They show the interesting result, that in general it is not possible to compute radial drawings of level planar graphs without crossings, no bends, and uniformly distributed vertices on the levels at the same time without violating the leveling. Further, they investigate tradeoffs by relaxing some aesthetic criteria to obtain drawability. Recently, Brandes and Pich [25] presented an energy based method to radially visualize undirected graphs. They extend stress majorization techniques [72] with a weighting scheme based on vertex distances that imposes radial constraints on the layout. Note, radial level drawings are different from circular drawings [16, 111, 134] where only one circle contains all vertices as described in Sect. 2.1.1.1 and do not comply with the radial drawings of [43, 44] where edges are drawn straight-line and level lines are not equidistant.

We present the first extension to the hierarchical framework for concentric level lines [2, 11, 12, 14]. All benefits, i. e., aesthetic criteria, of the traditional horizontal approach are preserved.

3.1 Cycle Removal

The problem is identical to the cycle removal in horizontal drawings. Thus, any of the methods described in Sect. 1.2.1 can be used.

3.2 Level Assignment

The basic problem for computing the level function $\phi : V \to \{1, \ldots, k\}$ for a DAG $G = (V, E)$ is the same as in horizontal drawings and any existing level assignment algorithm for horizontal leveling can directly be used for radial drawings. The optimization criteria slightly change, however: Radial drawings use k concentric circles to place the vertices of the k levels. Contrary to the constant line lengths in horizontal drawings, the perimeters of the circles grow proportionally longer with ascending level numbers: On an outer circle, there is space for more vertices than on an inner circle.

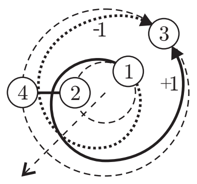
A straightforward idea is to apply the longest path level assignment method from Sect. 1.2.2 from outer to inner levels: First, each sink of the graph is assigned to the highest level. For the remaining vertices the level is recursively defined by $\phi(v) = \min\{\, \phi(w) \mid (v, w) \in E\,\} - 1$. This puts each vertex on the outermost possible level while minimizing the number of levels k. This is a rather simple approach, however, there is no explicit balancing of level sizes.

For a better vertex distribution, we extend the Coffman/Graham algorithm as described in Sect. 1.2.2 such that it takes into account the growing perimeter of the circles. Coffman and Graham compute a leveling where the number of vertices per level is bounded by a given global constant ω. We change this bound to be a function $\omega(i)$ which grows proportionally with the index $1 \leq i \leq k$ of a level, i. e., $\omega(i) \sim \omega \cdot i$.

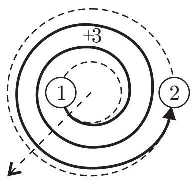
Similar to the horizontal framework, the level graph is normalized as the last step of the level assignment phase to make it proper. For drawing (radial) level graphs it is necessary to know where long edges should be routed, i. e., between which two vertices on a spanned level. Thus all long edges are subdivided in proper segments by up to $\mathcal{O}(k \cdot |E|)$ new dummy vertices. In the following, we will only consider proper k-level graphs $G = (V, E, \phi)$ if not mentioned otherwise.

3.3 Crossing Reduction

The problem of reducing crossings in radial drawings is more challenging compared to horizontal drawings, as it involves both vertex ordering and edge routing problems [88]. That is, even if the orderings of the vertices on two consecutive levels are fixed, we still need to decide how to route, clockwise or counter-clockwise, each edge between them around the center in order to minimize the number of crossings. For radial embeddings it is also necessary to know where the (w. l. o. g. counter-clockwise) orderings start and end on each level. Therefore, we introduce a *ray* that tags this borderline between the vertices, see Fig. 3.3(b) for an example. The ray is a straight halfline (or at least a monotone polyline) from the center to infinity between the vertices on each level with extremal positions. We call edges crossing the ray *cut edges*. In horizontal drawings of level graphs a crossing between two edges only depends on the orderings of the end vertices. In radial level drawings, however, it is also necessary to consider the direction and the multiplicity in which the edges are wound around the center, clockwise or counter-clockwise. We call this the offset $\psi\colon E \to \mathbb{Z}$ of an edge. Thereby, $|\psi(e)|$ counts the crossings of an edge $e \in E$ with the ray. If $\psi(e) < 0$ ($\psi(e) > 0$), e is a *clockwise* (*counter-clockwise*) cut edge, i. e., the sign of $\psi(e)$ reflects the mathematical direction of rotation, see Fig. 3.3. If $\psi(e) = 0$, then e is not a cut edge and thus needs no direction information.



(a) Edge $(1, 3)$ drawn counter-clockwise and clockwise (dotted)



(b) $\psi((1, 2)) = +3$

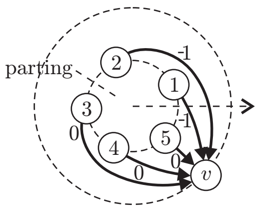
Figure 3.3. Offsets of edges

We define a *radial embedding* $\mathcal{E}$ of a proper graph $G = (V, E, \phi)$ to consist of the vertex ordering π and of the edge offsets ψ, i. e., $\mathcal{E} = (\pi, \psi)$. Compared to horizontal drawings there is an additional freedom in radial drawings without changing the number of crossings: *rotation* of a level i. A *clockwise rotation* moves the vertex v with minimum position on the ordered level $\phi(v) = i$ over the ray by setting $\pi(v)$ to the maximum on i. The other values of π are updated accordingly[1]. A *counter-clockwise rotation* is defined symmetrically. Rotations do not modify the "cyclic order", i. e., the neighborhood of every vertex on its radial level line is preserved. However, the offsets of the edges incident to v must be updated.

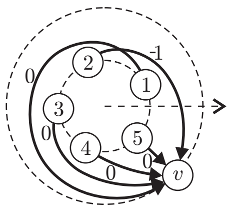
To reduce the number of crossings we first follow the conventional approach of the level-by-level sweep. The resulting problem is the $\mathcal{NP}$-hard *radial one-sided two-level crossing reduction* [2]. In a radial embedding $\mathcal{E}$ there can be more than one crossing between two edges $e_1, e_2 \in E_i$, $1 < i \leq k$, if they have very different offsets. Intuitively, this number is approximately equal to the difference of the offsets $|\psi(e_2) - \psi(e_1)|$. The exact formula shown by (3.1) is slightly different, however, with a small shift depending on the vertex ordering.[2]

$$\begin{aligned} \chi_{\mathcal{E}}(e_1, e_2) &= \max\left\{0, \left|\psi(e_2) - \psi(e_1) + \tfrac{b-a}{2}\right| + \tfrac{|a|+|b|}{2} - 1\right\}, \\ \text{where } a &= \operatorname{sgn}(\pi_1(u_2) - \pi_1(u_1)) \text{ and} \\ b &= \operatorname{sgn}(\pi_2(v_2) - \pi_2(v_1)) \end{aligned} \tag{3.1}$$

In [2] we have shown that a crossing minimal embedding implies $\psi(e_1) - \psi(e_2) \in \{0, 1\}$ for each pair of edges $e_1 = (u_1, v), e_2(u_2, v) \in E^-(v)$ with a common target vertex v and $u_1 \prec u_2$. That means the offsets differ at most by one. Thus, it is clear that only embeddings need to be considered, where there is a clear *parting* between all edges incident to the same vertex as in Fig. 3.4(a). The parting is that position of the edge list of v that separates the two subsequences with offsets ψ_0 resp. $\psi_0 + 1$. Otherwise unnecessary crossings are generated between the incident edges, e. g., see Fig. 3.4(b). In the remainder we only consider radial embeddings with small edge offsets -1, 0, and 1, because large offsets correspond to very long edges which are difficult to follow. Further, winding edges more than once around the center only would tend to increase its number of crossings.



(a) With parting



(b) Mixed

Figure 3.4. Not all offset combinations for edges $(\cdot, v) \in E^-(v)$ result in few crossings

[1]This can be done implicitly if using ordered lists for the vertices of each level.

[2]Although high offsets are never useful for a low number of crossings, we nevertheless provide the general result, not only to show that it also can be computed in constant time, but also as an interesting problem in itself.

Our results have been used by Hong and Nagamochi [86–88]. They present a 15 and a 4.3992 approximation algorithm to radial one-sided crossing minimization. The former has a running time of $\mathcal{O}(|E_i| \cdot \min\{|V_{i-1}|, |V_i|\} \cdot \min\{|V_{i-1}| \cdot |V_i|, |E_i| \log |V_i|\})$, whereas the latter in worst case needs time exponential in the maximum vertex degree of the input graph. In both cases, the radial problem is reduced to a classic one with horizontal levels, on which then the approximation algorithm of Nagamochi [116] is applied.

3.3.1 Cartesian Barycenter and Median

Now we directly attack the radial one-sided two-level crossing reduction problem with the simple heuristics from Sect. 1.2.3.1. We try to reduce crossings among radial edges between the fixed level $i-1$ and the permutable level $1 < i \leq k$ in the following. In the horizontal barycenter crossing reduction method for every vertex in V_i the average value of the positions of its neighbors in V_{i-1} is computed. Afterwards, V_i is sorted according to this values following the rule of thumb "shorter edges have less crossings than longer edges". With some restrictions, this method can be directly used to compute a radial embedding: The horizontal vertex ordering defines a radial vertex ordering, and all edge offsets are set to 0. This neglects the additional freedom of radial edge routing, however, and therefore introduces more crossings than necessary. Even worse, the result depends on the position of the ray.

We propose another approach which also follows the basic idea that every vertex should be close to the average position of its neighbors. However, we use the terms "average" and "position" in a geometric sense. We assume the vertices of the fixed level V_{i-1} to be uniformly distributed on a unit circle, according to the given ordering. This defines Cartesian coordinates $(x(u), y(u)) \in \mathbb{R}^2$ for each $u \in V_{i-1}$. Then we compute the *Cartesian barycenter* for each $v \in V_i$ using (3.2).

$$\mathrm{bary}(v) = \left(\frac{\sum_{u \in N^-(v)} x(u)}{|N^-(v)|}, \frac{\sum_{u \in N^-(v)} y(u)}{|N^-(v)|} \right) \tag{3.2}$$

If the Cartesian barycenter (or later the Cartesian median) is $(0,0)$, then any position is equipollent. Thus, we take a random one. Afterwards, we sort the vertices circularly around the origin, i. e., by the angles of $\mathrm{bary}(v)$ in polar coordinates. Finally, we distribute the vertices of level i uniformly on a concentric circle with radius 2 and choose for the offset of each edge one of -1, 0, or $+1$, whichever leads to the shortest edge in a geometric sense. The needed running time for one level is $\mathcal{O}(|E_i| + |V_i| \log |V_i|)$.

The *Cartesian median* heuristic is similar to the Cartesian barycenter heuristic. The only difference is that we take component-wise the x and y median instead of the component-wise barycenter. The running time stays the same, since $\mathrm{med}(v)$ can be computed in $\mathcal{O}(|N^-(v)|)$, see [37]. The median values depend on the underlying coordinate system (origin and rotation). But since we use the same coordinates for all median computations, this is no problem. Rotated coordinate systems, however, might lead to different results.

3.3.2 Radial Sifting

In contrast to the fast and simple algorithms described in the previous section, we now develop an extension of the sifting heuristic to radial levels. It is slower and slightly more involved but produces better results, i. e., which have fewer crossings. Recall that the same holds for traditional sifting on horizontal levels. As above we describe one-sided two-level crossing reduction between levels $i-1$ and i. Again we do not exploit the traditional sifting method using a crossing matrix but use the relative sifting approach as described in Sect. 2.1.1.2.

Unfortunately, we cannot directly use (1.3), because in radial sifting the crossing numbers also depend on the edge offsets, which are not constant in our approach. A change in the offset of an edge may affect all other edges instead of only the ones incident to the swap vertices. Therefore, the overall running time of this part of the algorithm for one sifting round is $\mathcal{O}(|E_i|^2)$ instead of $\mathcal{O}(|V_i| \cdot |E_i|)$ for horizontal relative sifting. The total running time of the algorithm, however, is dominated by the next step anyway.

In addition to the position of vertex u, we also have to compute the offsets of the incident edges. As the traversing vertex u moves along the i-th level circle in counter-clockwise direction, we update the offsets accordingly. Since there must be a clear parting of edges incident to u, we do not consider each possible offset combination for each (new) position of u. Intuitively, the parting of the edges should move around level $i-1$ in the same direction as u on level i, but on the opposite side of the circle. Otherwise, the edges incident to u get longer and tend to increase the number of crossings. Thus, we only decrease edge offsets by 1, starting with $\psi(e) = 1$ for all incident edges $e = (\cdot, u) \in E_i$, and we also do this one by one in the order of the end vertices on level $i-1$. The decision for which offsets are updated at which position of u is made subject to whether this leads to an improvement or not. Note that the parting may move around level $i-1$ twice, as offsets are decreased from 1 to -1. For one sifting round, we obtain an overall running time in $\mathcal{O}(|V_i|^2 \cdot |E_i|)$.

To allow a harmonic drawing of the computed embedding in the next phase a final postprocessing is useful which rotates level i with respect to uniform edge lengths. Since our algorithm starts with an offset of 1 for every edge and stops at the first best parting among several others which are as good, a straightforward drawing of the embedding is twisted too much (in counter-clockwise direction). However, this $\mathcal{O}(|E_i|)$ time consuming postprocessing step is only for aesthetic reasons and does neither affect the number of crossings nor the asymptotic running time.

3.3.3 Experimental Results

To analyze the performance of our one-sided two-level heuristics, we have implemented them in Java. Our experimental results show that all radial heuristics generate fewer crossings than their horizontal equivalents, experimentally by a factor of 0.7. This is a very encouraging result, since the running times of the radial algorithms (except sifting) are similar. As in the horizontal case [92], Cartesian barycenter on average leaves slightly fewer crossings than Cartesian median. Another similarity is that radial sifting is the best among all three radial heuristics, but also the slowest. Usually only few sifting rounds (3–5 for reasonable problem instances) are necessary to reach a local optimum for all vertices simultaneously, and the largest reduction of crossings usually occurs in the first round. In our experiments we further observed that the quality of radial sifting does not depend much on the quality of the initial embedding. However, a poor initialization increases the number of sifting rounds needed and thus raises the absolute running time. A complete implementation in JAVA sweeping with radial one-sided sifting level-by-level over k levels can be found in [85].

3.3.4 Radial Sifting with Intra-Level Edges

Similar to Sect. 2.1.1 we extend the sifting algorithm from Sect. 3.3.2 to *extended one-sided two-level radial sifting* of extended k-level graphs $G = (V, E, H, \phi)$ as sweep over the levels V_{i-1} (fixed) and V_i (permutable) with $1 < i \leq k$. Again the underlying minimization problem is $\mathcal{NP}$-hard. The resulting *extended radial levels drawings* are useful, e. g., for the visualization of social networks where the importance (centrality) of an actor (modeled by a vertex) defines its level [21–23]. More specifically, traditionally the visualization of an actor's status in a social network is a drawing where the levels are not equidistant, because each level represents a real-valued centrality index for the actors [26]. Since the centrality values often differ only marginally, status values can be clustered. The actors with centrality values in the same range are assigned to the same level to avoid perceptual problems of having too many levels. However, this approach introduces many intra-level edges. Another application of extended level graphs with radial drawings is the visualization of *micro/macro graphs* [17], e. g., arising from group analysis or role assignment in social networks [21]. In general, intra-level edges may help to gain better aspect ratios, since drawings tend to be much longer than wide, especially with the Sugiyama method.

We represent (except for the innermost level) intra-level edges using circular arcs with different radii in order to avoid overlapping edges and crossings between vertices and edges, see Fig. 3.5. The interpolation point in the middle is computed according to the edge lengths, i. e., the number of spanned vertices plus 1. Thus, the interpolation point of longer edges is closer to the concentric center and a crossing free nesting is possible. Contrary to straight-line edges but similar to the inter-level edges, there are two possibilities to wind the intra-level edges around the center, clockwise or counter-clockwise. For a low crossing number and low visual complexity, we always use the direction with shorter length, i. e., smaller or equal to $\lfloor \frac{|V_i|}{2} \rfloor$. As in Sect. 2.1.1, we restrict to draw the arcs only one-side of the level lines, say inside (above), in order to model the problem as in the classic Sugiyama framework.

Like in Sect. 2.1.1 we split the computation for the change in crossing numbers when swapping two vertices in the three values $\Delta = \Delta_E + \Delta_H + \Delta_{EH}$. Δ_E is computed exactly as discussed in Sect. 3.3.2.

For an easy notation $(v_1, v_2) \in H_i$ denotes an intra-level edge that is wound counter-clockwise around the center starting at vertex v_1 and ending at vertex v_2. This partitions

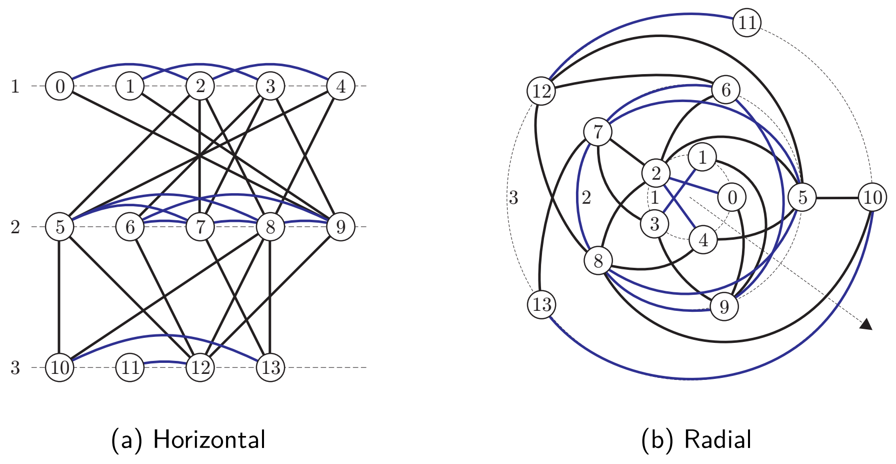
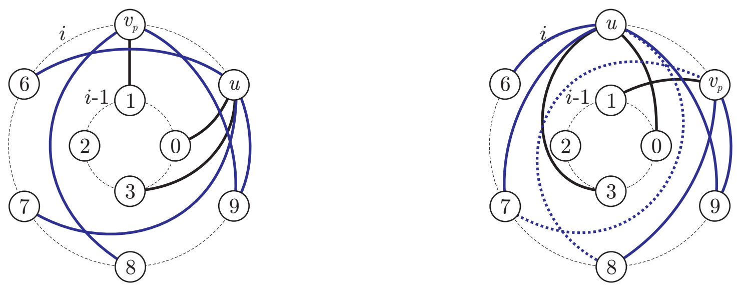


Figure 3.5. Example extended level drawings

the intra-level adjacency $H(v)$ for each $v \in V_i$ in two sets, the *incoming intra-level edges* $H_l(v) = \{(\cdot, v) \in H_i\}$ and the *outgoing intra-level edges* $H_r(v) = \{(v, \cdot) \in H_i\}$. Per convention, we store edges for which each direction results in the same length in H_l. $H_l(v)$ and $H_r(v)$ are kept sorted according to the $\prec_v$-ordering of the neighbors of v. During the sifting step, both the incoming and the outgoing intra-level adjacency lists of both vertices u and v_p are updated after their sifting swap. Then the computation of Δ_H is basically the same as in Sect. 2.1.1.2.

It remains the computation of Δ_{EH} for crossings between inter-level and intra-level edges. With the splitting of the intra-level adjacency for each vertex introduced in the previous paragraph, computing the change in crossings Δ_{EH} when swapping two consecutive vertices u and $v_p \in V_i$ stays principally the same as in (2.2) of Sect. 2.1.1. Thereby, we again neglect short edges $\{u, v_p\}$ which do not contribute to the crossings. What remains is the additional freedom of routing the intra-level edges around the center in two different directions. Contrary to crossings between only intra-level edges, this now has an effect as Fig. 3.6 shows.

To overcome this problem, we use the heuristic to always prefer the shorter direction. We denote intra-level edges that span at least $\lfloor \frac{|V_i|}{2} \rfloor - 1$ vertices as *long edges*. After the swap of the traversing vertex u with its (counter-clockwise) successor v_p, the length of all incoming intra-level edges of u, in $H_l(u)$ and all outgoing intra-level edges of v_p in $H_r(v_p)$ is increased by 1. Similarly, the length of all outgoing edges of u in $H_r(u)$ and all incoming edges of v_p in $H_l(v_p)$ is decreased by 1. In the case of an increase, it only can happen that the first (last) edge of the adjacency list $H_l(u)$ $(H_r(v_p))$ becomes a long edge. This is true, since we keep the adjacency lists ordered according to $\prec_u$ $(\prec_{v_p})$ and ascending edge lengths. One special case remains: If $|V_i|$ is even, then some intra-level edges may have the same length $\frac{|V_i|}{2}$ in both directions. We call them *vis-à-vis edges*, because they are incident to two vertices that are placed opposite to each other on their circular level. In order to locally minimize the number of crossings, we break ties in favor of the direction that causes less crossings with inter-level edges.



(a) Compute number of crossings before swap

(b) Update number of crossings after swap

Figure 3.6. Crossings between inter-level and intra-level edges and optimal routing

We obtain an overall time complexity of $\mathcal{O}(|V_i|^2 \cdot (|E_i| + |H_i|))$ for one round of extended radial sifting.

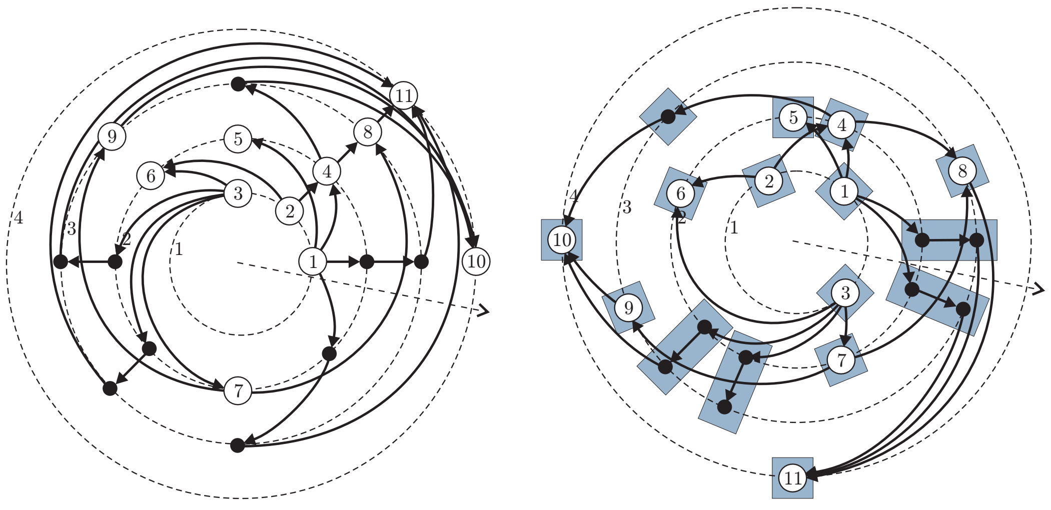
3.3.5 Global Sifting

To achieve all the benefits of our global sifting approach presented in Sect. 2.1.2, we now generalize it to work on not necessarily proper radial level graphs $G = (V, E, \phi)$ such as the one shown in Fig. 3.7(a). The algorithm guarantees radially aligned inner segments of long edges out of the box and can be used directly with only minor modifications: Each block of the block list $\mathcal{B}$ has its own angle. See Fig. 3.7(b) for an example. As above we introduce a ray to imply an ordering to $\mathcal{B}$ and use the offsets as defined for each outer segment. When sifting a traversing block $A \in \mathcal{B}$, we have to update two partings. These are the two borders between the counter-clockwise and clockwise segments on the levels above and below A. More precisely, the partings are among edges incident either to the topmost or to the bottommost vertex of A, respectively, see Fig. 3.8.

Since we can do the parting updates independently of each other and add the results of the changes in crossing counts to our Δ-value in relative sifting, we can use the same technique as in Sect. 3.3.2: We sift a block from its current position in counter-clockwise direction. Thus, for few crossings the partings have to follow in this direction on their levels. The test during the swap whether changing the orientations of some of the first of the (ordered) incident segments of A by incrementing their offsets, and thus putting them last, leads to less crossings and counting the difference raises the overall running time to $\mathcal{O}(|E|^3)$. However, the subsequent coordinate assignment described in Sect. 3.4 relies on the absence of type 2 conflicts which is up to now only guaranteed here.

The leftmost alignment[3] of a level (pushing the vertices of a level clockwise towards the ray without changing their ordering) or the distribution of the vertices equidistant on their levels under maintenance of the topology (and the crossings as they are) may

[3]Remember, an embedding is defined by positions of vertices in ordered levels (and edge offsets) and not by (absolute) coordinates.


(a) Input (b) Unique block angles

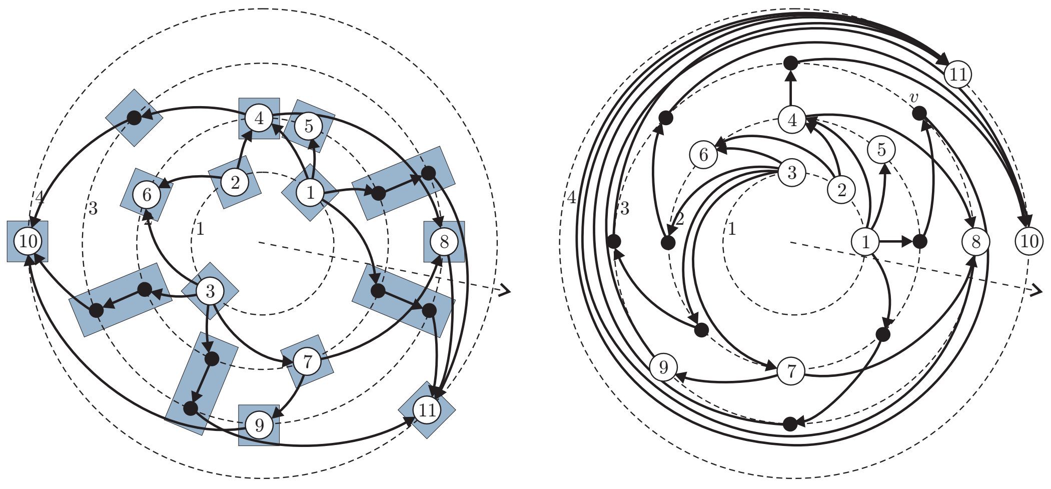

(c) After global sifting (d) Resulting Embedding

Figure 3.7. Global crossing reduction for a radial drawing

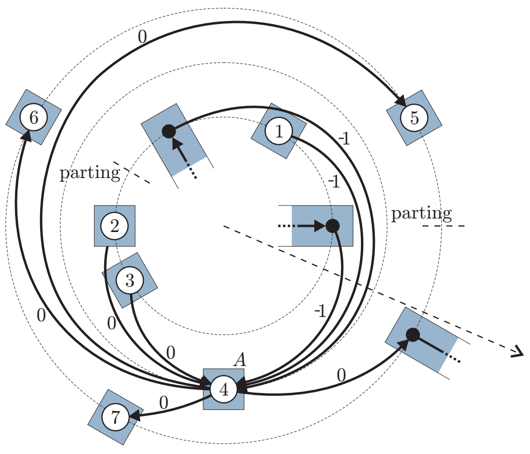


Figure 3.8. Partings of the block A in a radial drawing

make some edges longer, see Figs. 3.7(c) and (d). In worst case, some edges may span afterwards up to exclusively $720°$. However, since no vertex is pushed over the ray all edge offsets will remain valid and stay at most 1, which is a precondition of the coordinate assignment phase described next. For example, in Fig. 3.7(d), the edge $(v, 11)$ spans $360°$ instead of $0°$ and being straight. This avoids three new crossings.

3.4 Coordinate Assignment

In radial level drawings we draw the edge segments as segments of spirals, unless they are radially aligned and, thus, are drawn as straight lines. This results in strictly monotone curves from inner to outer levels and ensures that segments do not cross inner level lines or cross each other unnecessarily. This fourth phase of the framework is usually constrained not to change the vertex orderings computed previously, what is especially useful if the input embedding is a planar embedding, e. g., as generated by [1, 7]. Further, the drawing algorithm should support commonly accepted criteria for readability and aesthetics, like small area, good separation of (dummy) vertices within a level, length and slope of edges, straightness of long edges, and balancing of edges incident to the same vertex. In our opinion edge bends in radial level drawings tend to be even more disturbing than in horizontal level drawings because of the visual edge complexity of spiral segments. Thus, we again base our algorithm on the approach of Brandes/Köpf [24] as described in Sect. 1.2.4 which guarantees at most two bends per edge. Further it prioritizes vertical alignment, which helps us to obtain radial alignment. The criterion of small area in horizontal coordinate assignment, i. e., to obtain small width, turns to uniform distribution of the vertices on the radial levels. As a consequence, a user defined parameter δ for the minimum vertex separation as used in Sect. 1.2.4 is not needed. Since the (proper) input embedding for this phase maintains the position $\pi(v)$ for every vertex $v \in V$, the position of the ray is implicitly evident, i. e., on each level it lies between the two vertices with extremal positions.

3.4.1 Preprocessing

If an inner segment is a cut segment in the input embedding, i. e., if it crosses the ray, then the maximum of two bends for the corresponding long edge cannot be guaranteed, see Fig. 3.9 for an example. We call this situation a *type* 3 *conflict* as a generalization of the notion in [24].

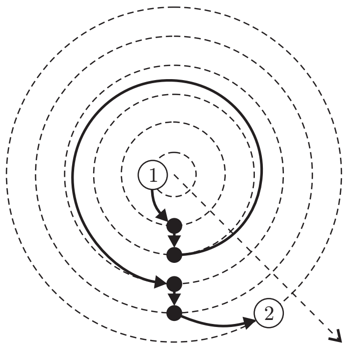


Figure 3.9. Type 3 conflict

A simple solution is to demand the absence of inner cut segments in the input, similar to type 2 conflicts in [24] and here. A different, more constructive, and always doable approach described next is to *unwind* the graph and rotate some levels. This changes the position of the ray, i. e., the offset of some edges, and thus the embedding. However, this does not affect a later drawing, since the relative orderings of the vertices stay the same.

Clearly, levels $3 \leq i < k$ with incoming inner segments are unwound one after the other by rotating the whole respective outer graph, i. e., all levels $\geq i$ are rotated simultaneously by multiples of $360°$ for each i. This is done in such a way, that the resulting offsets of all inner segments are either 0 or $+1$ and that there are no two dummy vertices $v_1, v_2 \in V_i$ on the same level i with $\psi\big((u_1, v_1)\big) = 0$, $\psi\big((u_2, v_2)\big) = +1$, and $v_1 \prec v_2$ for any pair of dummy vertices $u_1, u_2 \in V_{i-1}$ on the previous level. This is always possible in linear time as constructively shown in [2, 12]. Up to now the position of the ray and the orderings of the vertices remained unchanged. Finally, we use standard rotations (Sect. 3.3) to eliminate the remaining crossings of inner segments with the ray. Recall, rotation of a single level i is different from rotating levels during unwinding. Here we do not rotate by (multiples of) $360°$ in general and do not rotate all levels $\geq i$ simultaneously. Clearly, we rotate each level clockwise such that the ray enters the position after the last dummy vertex incident to an inner segment with an offset of originally $+1$. In the end, all inner segments have an offset of 0. However, the approach may create many crossings of outer segments with the ray.

3.4.2 Intermediate Horizontal Layout

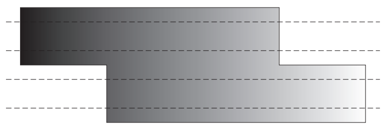
In the next step we generate an *intermediate horizontal drawing* of the radial level embedding with the Brandes/Köpf algorithm of Sect. 1.2.4. Therefore, we ignore all cut segments. Since the embedding is now free of type 3 conflicts, all inner segments of an edge are aligned vertically. The resulting layout will later be transformed into a concentric layout by concentrically connecting the ends of the horizontal level lines with their beginnings. Since the circumferences of circular level lines grow with ascending level numbers $1 \leq i \leq k$, we use a minimum vertex separation distance $\delta_i = \frac{1}{i}$ for each horizontal level i, which is in each case indirectly proportional to i. Hence, we obtain a uniform minimum arc length between two consecutive vertices on every level line with the radial transformation described in the next section.

3.4.3 Radial Layout

At this stage every vertex $v \in V$ has Cartesian coordinates $\big(x(v), y(v) = \phi(v)\big) \in \mathbb{R} \times \mathbb{N}$. For the transformation into a *radial 2D drawing* we interpret these coordinates as polar coordinates and transform them with (3.3) into Cartesian coordinates. The position of the ray denotes $0°$.

$$\big(x_{\text{2D}}(v), y_{\text{2D}}(v)\big) = \Big(y(v) \cdot \cos\big(\tfrac{2\pi}{z} \cdot x(v)\big), y(v) \cdot \sin\big(\tfrac{2\pi}{z} \cdot x(v)\big)\Big) \in \mathbb{R}^2 \tag{3.3}$$

The factor $\frac{2\pi}{z}$ normalizes the length of the horizontal level lines to the circumferences of the radial level lines. We set z as the largest horizontal distance between two vertices on the same level i plus δ_i. This normalization automatically realizes the necessary overlap between the left and the right contour of the horizontal layout when drawn radially, see Fig. 3.10 for a schematic illustration.


(a) Horizontal

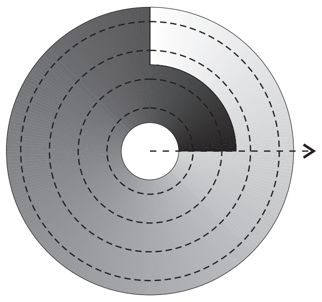

(b) Radial

Figure 3.10. Overlap of the left and right contour

After drawing the vertices, we draw each short edge $e = (u, v) \in E$ as a segment of a spiral. For that, we use the interpolation points p controlled by the parameter $t \in [0; 1]$ in (3.4). Since e can wind multiply around the center, we add $\psi(e) \cdot z$ to the x-coordinate of v. To obtain smooth edges, the number of interpolation points needed for drawing the edges with an approximating polyline or spline depends on the edge length and a quality factor.

$$\begin{aligned}\big(x_{\text{2D}}(p), y_{\text{2D}}(p)\big) =& \big((1-t)\cdot y(u) + t\cdot y(v)\big)\\ &\cdot\Bigg(\cos\left(\tfrac{2\pi}{z}\cdot\Big((1-t)\cdot x(u) + t\cdot(x(v)+\psi(e)\cdot z)\Big)\right),\\ &\quad\sin\left(\tfrac{2\pi}{z}\cdot\Big((1-t)\cdot x(u) + t\cdot(x(v)+\psi(e)\cdot z)\Big)\right)\Bigg)\end{aligned} \tag{3.4}$$

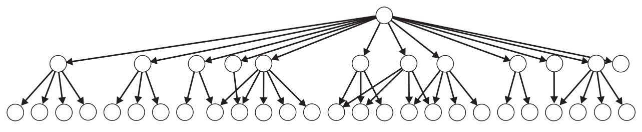
For generating a *radial 3D drawing* on the cylinder we use (3.5), where r_ω is the radius of the cylinder. Edges are interpolated similar to above.

$$\begin{aligned}\big(x_{\text{3D}}(v), y_{\text{3D}}(v), z_{\text{3D}}(v)\big) =&\\ &\left(r_\omega\cdot\cos\left(\tfrac{2\pi}{z}\cdot x(v)\right), r_\omega\cdot\sin\left(\tfrac{2\pi}{z}\cdot x(v)\right), y(v)\right) \in \mathbb{R}^2\times\mathbb{N}\end{aligned} \tag{3.5}$$

For the radial coordinate assignment we obtain a linear running time $\mathcal{O}(|V| + |E|)$ in the size of a proper level graph $G = (V, E, \phi)$.

3.4.4 Example Outputs

In the following we oppose some traditional layouts with their radial pendants generated with the JAVA implementation of [66]. Apart from the edge orientations, they share a common embedding, respectively. Note, all size indications on vertices include dummy vertices and references to the number of edges count proper edges, i. e., the segments.


(a) Horizontal

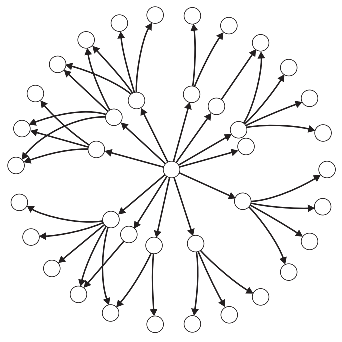
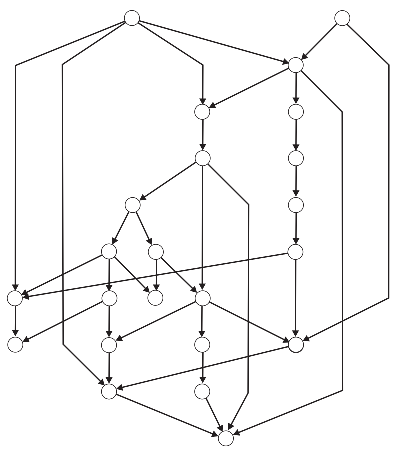

(b) Radial

Figure 3.11. Example with $|V| = 39$, $|E| = 47$, and $k = 3$


(a) Horizontal

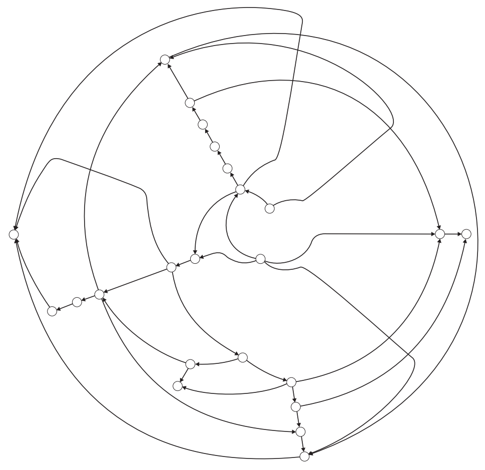
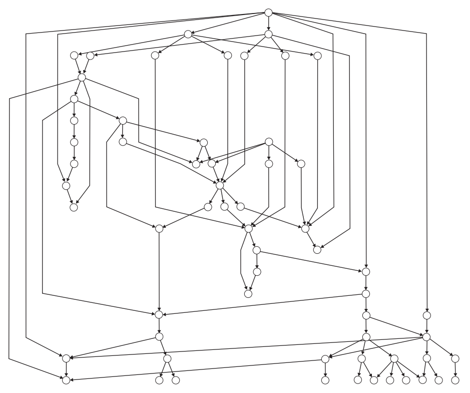

(b) Radial

Figure 3.12. Example with $|V| = 58$, $|E| = 69$, and $k = 10$


(a) Horizontal

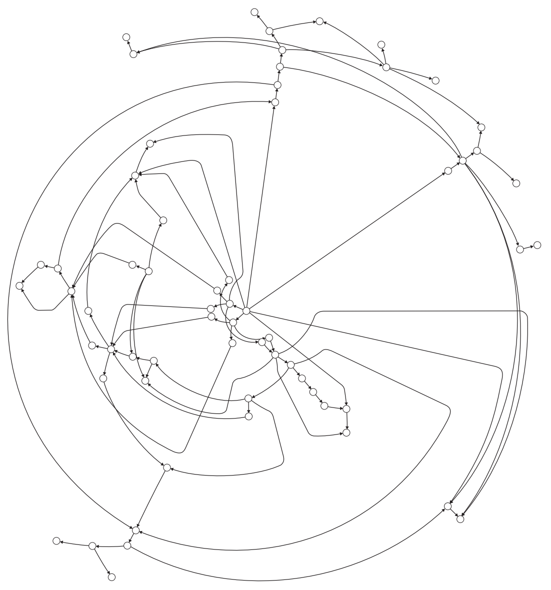
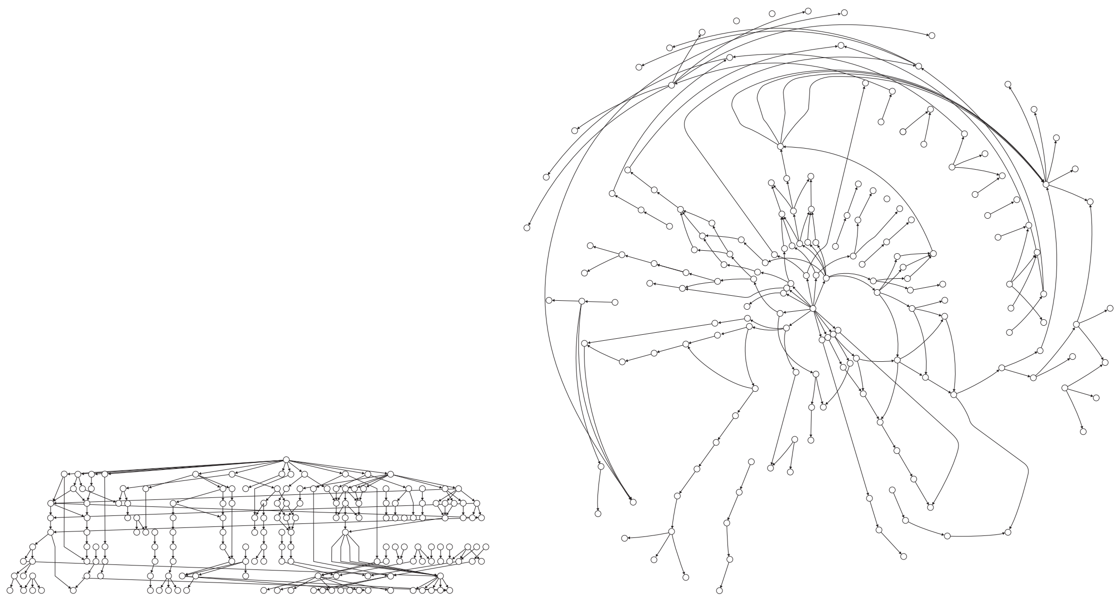

(b) Radial

Figure 3.13. Example with $|V| = 201$, $|E| = 228$, and $k = 18$



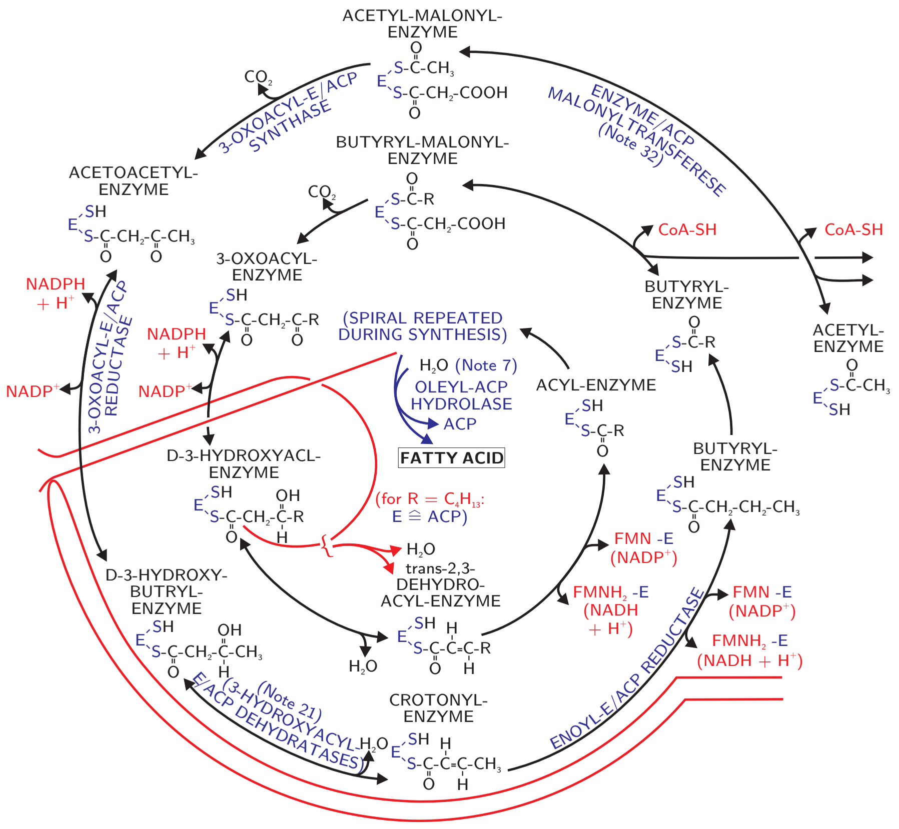
(a) Horizontal (b) Radial

Figure 3.14. Example with $|V| = 216$, $|E| = 230$, and $k = 10$

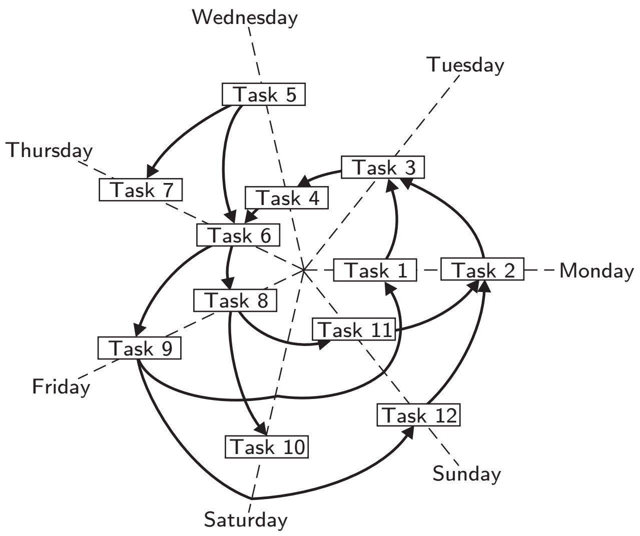
4 Cyclic Drawings

In the general case, the Sugiyama framework destroys cycles in its first phase by removing or reversing some edges, see Sect. 1.2.1. It turned out that this is very useful to draw directed graphs with no or only few cycles to visualize the common direction of information flow in diagrams. In some applications, however, it is not appropriate or even inacceptable to assume that there is some reasonable partial order in the graph. Then cycles should not just be considered as temporary "error" since they carry essential structural information which needs to be clearly visible or explicitly highlighted in a drawing. This forbids the removal of cycles. Examples are distinguished cycles in the life sciences like the fatty acid synthesis illustrated in Fig. 4.1(a) or the citrate cycle [110], and processes in a schedule which repeat in a daily or weekly turn [132], e. g., see Fig. 4.1(b). The users in this areas are accustomed to see the cycles as such, although drawn by hand up to now. Even more, these cycles often serve as a landmark [110]. In the area of information visualization, circular parallel coordinates are an adequate instrument to depict multi dimensional data in two dimensions [38, 65]. Cyclic drawings visualize data with a similar scheme.

Obviously, there are many situations where the destruction of cycles by reversing some edges to compute a layout and an ex post correction to their primal directions is not appropriate. This fact was already known to Sugiyama et al., so they proposed in their seminal paper from 1981 [138] beside the hierarchical framework a drawing style for cyclic graphs, which they called *recurrent hierarchies*. A recurrent hierarchy is a level graph with additional edges from the last to the first level. Here, two drawings are natural: The first is a 2D drawing, where the levels are rays from a common center forming a star, and are sorted counter-clockwise by their number, see Fig. 4.2(a). All vertices of one level are placed at different positions on their ray and an edge $e = (u, v)$ is drawn as a monotone counter-clockwise curve, i. e., as segment of a spiral, from u to v wrapping around the center at most once. The second is a 3D drawing on a



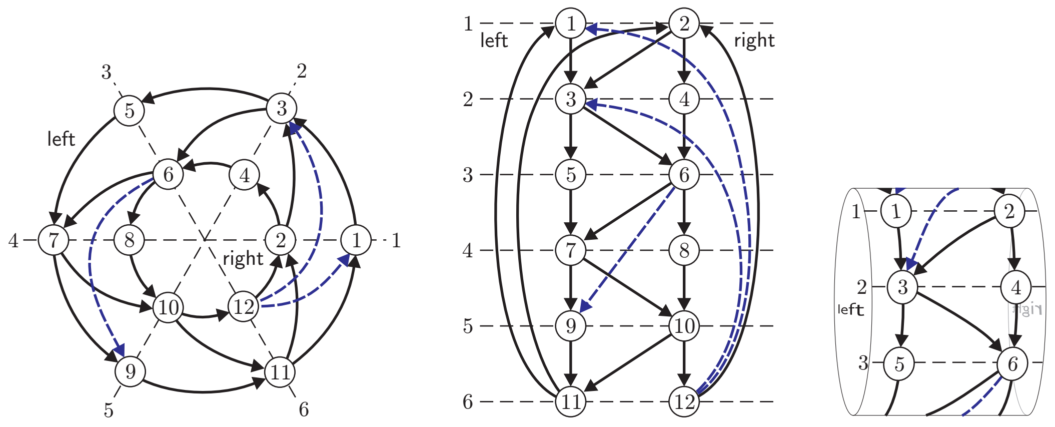
(a) Fatty acid synthesis [110]



(b) Schedule for repeating tasks. The edges illustrate (cyclic) dependencies on results of previous tasks

Figure 4.1. Cyclic level drawings in life sciences and scheduling

cylinder, see Fig. 4.2(c). In both cases, the last level k is a predecessor of the first level 1. As a consequence, the span of an edge $e = (u, v)$ is now (re-)defined as $\mathrm{span}(e) \coloneqq \phi(v) - \phi(u)$ if $\phi(u) < \phi(v)$, and $\mathrm{span}(e) \coloneqq \phi(v) - \phi(u) + k$ otherwise.

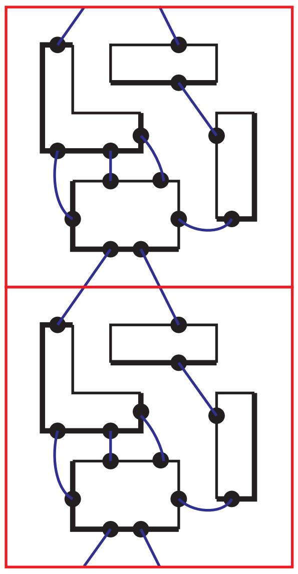


(a) Cyclic (b) Horizontal (c) Cyclic 3D

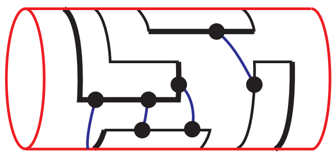
Figure 4.2. Example drawings

Typical applications for cyclic layouts can also be found in VLSI design. There, it is often necessary to build regular layouts of one dimensional arrays of identical cells as Fig. 4.3(a) exemplifies. Let S be the cell which is replicated, for example a bit or word on a memory chip. Then, the content and the links of S have to be layouted in such a way that it can be used to seamlessly tile an infinite strip. This is topologically nothing else than cylinder drawings of S as Fig. 4.3(b) shows, or the flat drawings as a star. We will see in Chapter 5 how to tile the plane with two dimensional arrays.
Recurrent hierarchies are known to most graph drawers – but unnoticed. A planar recurrent hierarchy is shown on the cover of the text book by Kaufmann and Wagner [96]. There it is stated that recurrent hierarchies are "unfortunately [...] still not well studied" [96, page 119]. One reason is that they are much harder. Intuitively, there is no start and no end, there are no top and bottom levels.

Sugiyama and Misue [137] have later introduced an alternative approach to the framework. They proposed a set of modifications of force-based algorithms to enforce cyclic orientation. In this context they ask [137, page 372]: "Can directed graphs with cycles be drawn in a way that it is easy to grasp the global flow of the graphs and the existence of cycles?" They use a concentric force field which rotates around the center and takes edges along. They report about promising experimental results for small example graphs, although they neglect given hierarchical dependencies. Their method is quite intuitive and easy to implement, however, as is common to force-based methods, highly suspectible to local minima and sensitive to the choice of the initial configurations and parameters. Recently, Pich [121] presented a method using spectral graph layout techniques [97] to generate cyclic drawings of directed graphs. However, this does not



(a) One dimensional array of cells [109]



(b) Cell representation on the cylinder

Figure 4.3. Applications in VLSI design

result in discrete levels for the vertices. Thus, we do not follow these approaches and enhance the hierarchic framework for cyclic level lines [3–5].

Clearly, cyclic drawings are only superior to traditional hierarchic drawings on graphs with a mainly cyclic rather than a hierarchic structure. Hypothesis about that can be analyzed with [120, 121] using spectral decompositions [97] of the *skew symmetric adjacency matrix* before deciding how to draw a given graph.

4.1 Cycle Removal

In cyclic drawings edges are irreversible and cycles are represented in a direct way. Thus, the cycle removal phase disappears from the common Sugiyama framework. This saves much effort, since the underlying problem is the $\mathcal{NP}$-hard feedback arc set problem [75]. Another advantage are short edges. The sum of the edge length can be smaller than in the hierarchical case: Consider a cycle consisting of three vertices. The only way to draw this graph in the Sugiyama framework is to reverse one edge which will span two levels. Therefore, the sum of the edge length will be four. In the cyclic case this graph can be drawn on three levels such that each edge has span one.

Note that any Sugiyama drawing is a cyclic Sugiyama drawing which discards the option to draw edges between the last and the first level. Due to this trivial reason, all benefits of such drawings exist in the cyclic case as well. However, the sum of the edge length and the number of crossings will often be smaller.

4.2 Level Assignment

Cyclic level assignment is akin to the NP-hard *directed circular arrangement problem*, where the vertices of a directed graph are embedded onto evenly spaced points on a circle such that the total (weighted) length of the counter-clockwise edges is minimized [71, 103, 104, 117]. This translates to an injective leveling, where each vertex gets it own cyclic level. A typical application for directed circular arrangement is to define the best periodical order of server broadcasts of data packets, where the edges represents (e. g., time or data) dependencies between the packets [104]. With cyclic leveling there is a generalization such that more than one packet can be broadcasted at a particular time step, e. g., over different (logical) channels. However, contrary to common circular arrangement methods, we do not consider so called *multicasts*, i. e., duplicating data packets within one broadcast round. Another main application of circular arrangement can be found in load minimization in directed ring networks like in Cisco's Dynamic Packet Transport technology [117]. The authors of [71] show that directed circular arrangement has no polynomial time approximation scheme (PTAS). Hence, it is likely that circular leveling admits no PTAS as well. For the theoretical background of cyclic arrangement of directed graphs see [71, 105].

Before we start with constructive algorithms to assign vertices to cyclic levels, we first work out the objectives for cyclic leveling and compare their complexities to the hierarchical style.

4.2.1 Complexity

There is a substantial and diametric discrepancy between the hierarchic and the cyclic leveling. Simultaneously minimizing the height and the width is $\mathcal{NP}$-hard in the former, cf. Sect. 1.2.2, and polynomial in the latter case. In fact for cyclic leveling this is even trivial: An edge $e = (u, v)$ does not impose any constraint on the leveling of the vertices u and v. Therefore, filling up an arbitrary height times width grid with vertices on arbitrary positions gives a compact leveling, since dummy vertices of long edges are traditionally not counted for the width, see Sect. 1.2.2. However, in general the resulting drawings are ugly and confusing.

Conversely, minimizing the sum of the edge lengths drawing on $k \geq 2$ levels is polynomial hierarchically, see Sect. 1.2.2, and is $\mathcal{NP}$-hard on cyclic levels. There the span can no longer be formulated by a system of efficiently solvable linear equations, because a case differentiation or the modulo operation is needed. We have shown the $\mathcal{NP}$-completeness by reductions for the *maximum bipartite subgraph* and the *graph k-colorability problems* [5]. Thus for the desired goal of an overall small span we have to use heuristics. Known approaches from the hierarchical case like the longest path method [96] or the Coffman-Graham algorithm [36] cannot be easily adapted to the cyclic case. They heavily rely on the fact that the graph is acyclic and start the leveling process at source vertices with no incoming edges. As it is not guaranteed that such vertices exist in the cyclic case at all, we introduce three new heuristics.

4.2.2 Breadth First Search

The BFS-heuristic to level a graph (V, E) to obtain a k-level graph $G = (V, E, \phi)$ with maximum width ω is rather simple: We choose an arbitrary start vertex $v \in V$, set $\phi(v) = 1$ and perform a directed BFS from v. Define $\operatorname{next}(i) := (i \mod k) + 1$ to be the cyclic next level after level $1 \leq i \leq k$. Whenever we reach a vertex w for the first time using an edge (u, w), we set $\phi(w) = \operatorname{next}(\phi(u))$ if this level does not contain ω vertices already. Otherwise, we move w to the first non-full level. Consequently, the tree edges will have a rather short span. However, the back edges are not taken into account for the leveling at all and may get arbitrarily long with a span up to k, thus. The algorithm needs $\mathcal{O}(|V| + |E| + k^2)$ time.

4.2.3 Minimum Spanning Tree

The next heuristic has similarities to the algorithm of Prim [37], which computes the *minimum spanning tree* (MST) of a graph. We sequentially level the vertices by a greedy algorithm. Let $V' \subset V$ be the set of already leveled vertices. Whenever we level a vertex $v \in V - V'$, all incident edges to vertices in V' get a fixed span. Therefore, we set $\phi(v)$ such that the sum over this spans is minimized. Note that there are possibly more edges incident to v which are also incident to not yet leveled vertices in $V - V'$. These edges will be considered when their second end vertex gets leveled. We decide in which order to add the vertices by using a distance function $\delta_{\mathsf{X}}(v)$, for which we discuss four options. Thereby, let $E(v, V') := \{\, \{v, w\} \in E \mid w \in V' \,\}$ be the set of (underlying undirected) edges incident to v and an already leveled vertex w and let $\operatorname{span}(E^*) := \sum_{e \in E^*} \operatorname{span}(e)$ for $E^* \subseteq E$ be a shortcut for the sum over the spans of the edges in a set E^*.

Minimum Increase in Span We choose the vertex which will create the minimum increase in span in the already leveled graph:

$$\delta_{\mathsf{MIN}}(v) = \min_{\phi(v) \in \{1, \ldots, k\}} \operatorname{span}(E(v, V')) \tag{4.1}$$

Minimum Average Increase in Span The distance function δ_{MIN} will place vertices with a low degree first, as vertices with a higher degree will almost always cause a higher increase in span. Therefore, considering the increase in span per edge is reasonable:

$$\delta_{\mathsf{MIN\_AVG}}(v) = \min_{\phi(v) \in \{1, \ldots, k\}} \frac{\operatorname{span}(E(v, V'))}{|E(v, V')|} \tag{4.2}$$

We distribute isolated vertices uniformly on the non-full levels in the end.

Maximum (Average) Increase in Span Choose the vertex which causes the maximum (average) increase in span per edge:

$$\delta_{\mathsf{MAX}}(v) = \frac{1}{\delta_{\mathsf{MIN}}(v)}, \qquad \delta_{\mathsf{MAX\_AVG}}(v) = \frac{1}{\delta_{\mathsf{MIN\_AVG}}(v)} \tag{4.3}$$

The reasoning behind is that a vertex which causes a high increase in span will cause this increase when leveled later as well. But if we level the vertex now, we can possibly level other adjacent, not yet leveled vertices in a better way. A similar idea could be to simply choose a vertex with a high degree, i. e., with many not yet leveled neighbors.

Note that we use $\delta_{\mathsf{X}}(v)$ only to determine which vertex is leveled next. After we have decided which vertex v to take next, we assign v to level $\phi(v)$ such that the increase in span will be minimized. In some cases several levels for v will create the same increase in span. Then we choose a level for v which minimizes $\sum_{e \in E(v,V')} \operatorname{span}(e)^2$ as well. Thus, v gets a level which is more centered between v's leveled neighbors and the resulting drawing is more balanced. In each case, we can only use a level which has not yet the maximum number ω of vertices on it. Vertices with already leveled neighbors always block a place on their optimal level such that they can later be placed on the level.

The heuristic needs $\mathcal{O}(|V| \log |V| + k \cdot \deg(G) \cdot |E|)$ time. According to our experiments [5, 106], all MST variants do not differ very much in quality, i. e., in the overall span. However, to take the minimum average increase in span as distance function seems to be the best.

4.2.4 Force Based

Spring embedders use a physical model to simulate the edges as springs [70, 94, 96]. Forces between vertices are computed and the vertices are moved accordingly. Transferring this idea to the cyclic leveling problem, we could use a force function similar to conventional energy based placement algorithms as follows:

$$\operatorname{force}(v) = \sum_{(v,w) \in E} (\operatorname{span}((v,w)) - 1)^2 - \sum_{(u,v) \in E} (\operatorname{span}((u,v)) - 1)^2 \tag{4.4}$$

However, moving a vertex to its energy minimum using this force will not minimize the span of the graph [5], i. e., (4.4) minimizes the deviation between the edge lengths. Furthermore, the span may increase when moving a vertex towards its energy minimum, as some edges can flip from span 1 to span k. We solve this problem by using directly the span as the (undirected) force which is minimized, see (4.5).

$$\operatorname{force}(v) := \operatorname{span}(E(v,V)) \tag{4.5}$$

We move the vertex with the maximum impacting force. And we directly move the vertex to its energy minimum, which is the level such that the span is minimized. For this, we test all possible (non-full) levels. Note that moving all vertices at once as done in [94] would not decrease the time complexity here.

As an initial leveling we use the result of the minimum average increase in span tree heuristic from Sect. 4.2.3. For each iteration we need $\mathcal{O}(|V| \log |V| + k \cdot |E|)$ time. Experiments [5, 106] confirm our expectation that this heuristic is the best among all our cyclic leveling strategies.

4.3 Crossing Reduction

As it is the case with radial drawings, the cyclic style reduces the number of crossings compared to the horizontal style in general. See Figs. 4.2(a) and (b) for an example. At the threshold with no crossings [9], there are cyclic level planar graphs which are not level planar. See Fig. 4.2 by considering the solid edges only.

Our global sifting algorithm from Sect. 2.1.2.1 can be used directly for cyclic level graphs without major changes. We put each block to an unique radius around the center rather than on an own x-position, i. e., the positions in the block list $\mathcal{B}$ define these radii. See Fig. 4.4 for an example. Recall that there are no "ring blocks" since every long edge is wrapped around the center at most once and, thus, each block spans at most $k-2$ levels. We obtain the same time complexity $\mathcal{O}(|E|^2)$ for one sifting round in the block graph for a not necessarily proper k-level graph $G = (V, E, \phi)$ which is independent from the number of dummy vertices. A difference to horizontal global sifting is that up to four level pairs must be considered for a change in crossings per swap, since two blocks may have an overlap which builds up at most two disjoint intervals. See Fig. 4.5 for an example. The algorithm shown as pseudo code in [4] is already aware of this. For results of experiments with an implementation in JAVA substantiated by benchmarks see [81].

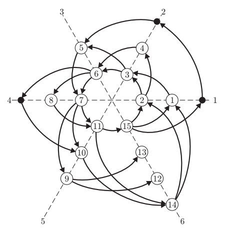
Note that one-sided two-level algorithms cannot be applied here, since each of them pushes most of the crossings to the next level only. There is no possibility to push the crossings "out of the drawing" as it can be done with horizontal level lines. Since the (last) cyclic sweep has to stop at some time at a concrete level, it would leave many crossings between this level and its successor level. Furthermore, the absence of type 2 conflicts cannot be guaranteed then, even if using the simple barycenter or median heuristics which avoid them on horizontal levels. Again, this will be necessary for the coordinate phase as introduced in Sect. 4.4. See [9, 10] for the special case of cyclic level planar graphs, which admit crossing free embeddings.

4.4 Coordinate Assignment

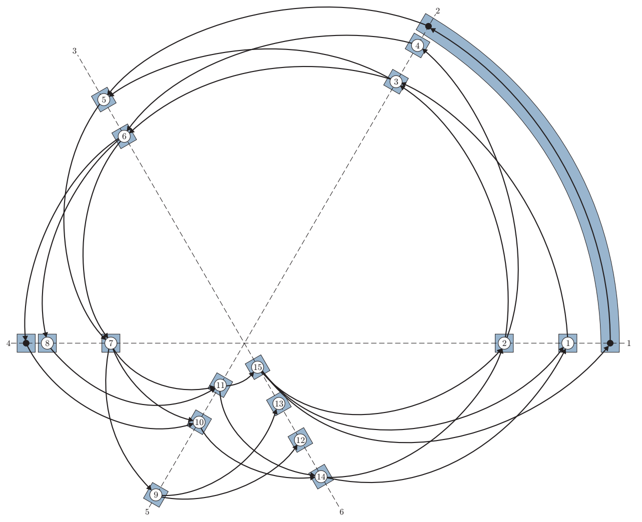
Once again we modify the algorithm of Brandes and Köpf from Sect. 1.2.4, now for cyclic k-level graphs $G = (V, E, \phi)$ to provide a linear time algorithm using quadratic area with at most two bends per edge. We assume that the input embedding is proper, i. e., V includes dummy vertices if necessary, and does not contain type 2 conflicts.

We represent drawings of cyclic level graphs in an *intermediate drawing* in the remainder of the section assigning each vertex v two coordinates $x(v) \in \mathbb{R}$ and $y(v) = \phi(v) \in \mathbb{N}$. The x-coordinate increases from *left* to *right* (by at least unit length between successive vertices), the y-coordinate increases *downwards* in edge direction, see Fig. 4.6(b). All vertices on level 1 are duplicated on level $k+1$ using the same x-coordinates. Each segment $(u, v) \in E$ is drawn straight-line from $\big(x(u), y(u)\big)$ to $\big(x(v), y(u)+1\big)$ with *slope* $\frac{1}{x(v)-x(u)}$.

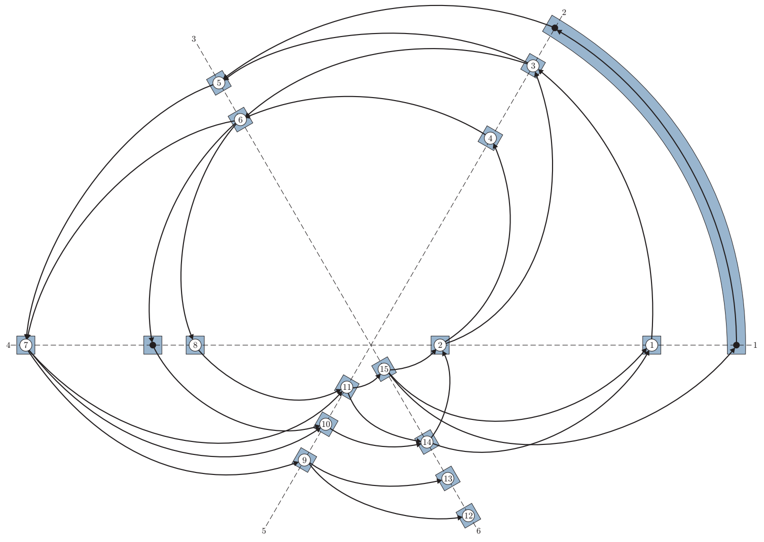
A *cyclic 2D drawing* as shown in Fig. 4.6(a) is obtained from an intermediate drawing by transforming each point $p = \big(x(p), y(p)\big)$ of the plane by (4.6), with the radius



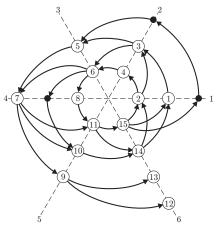
(a) Input



(b) Unique radii for the blocks



(c) After global sifting



(d) Resulting Embedding

Figure 4.4. Global crossing reduction for a cyclic drawing

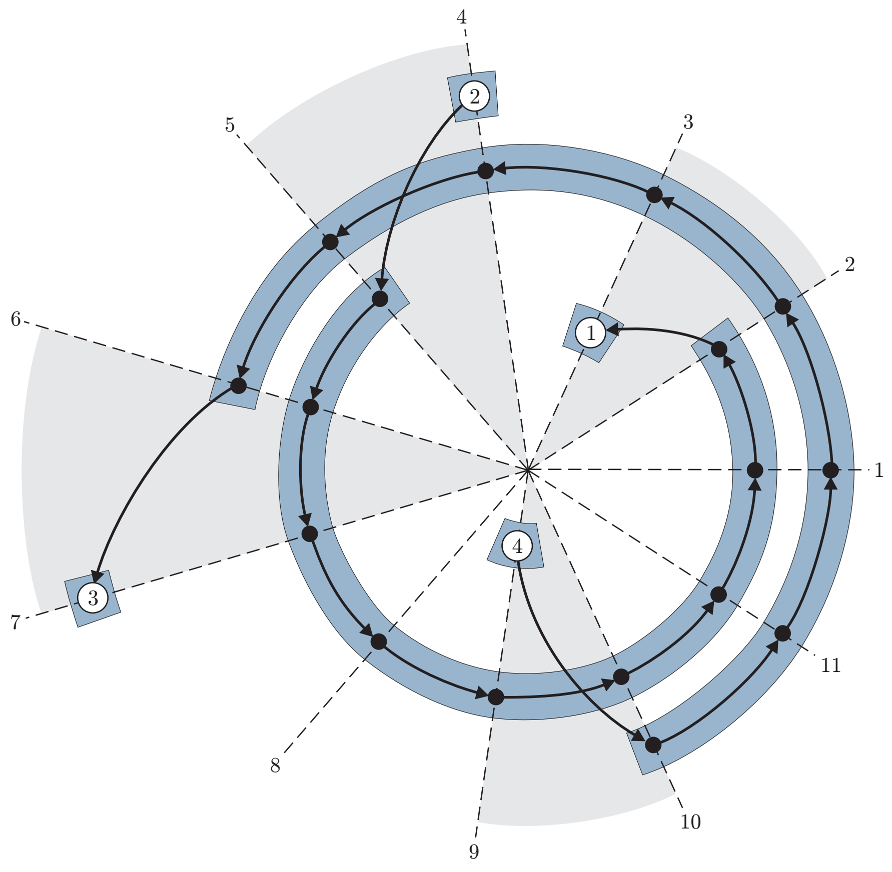
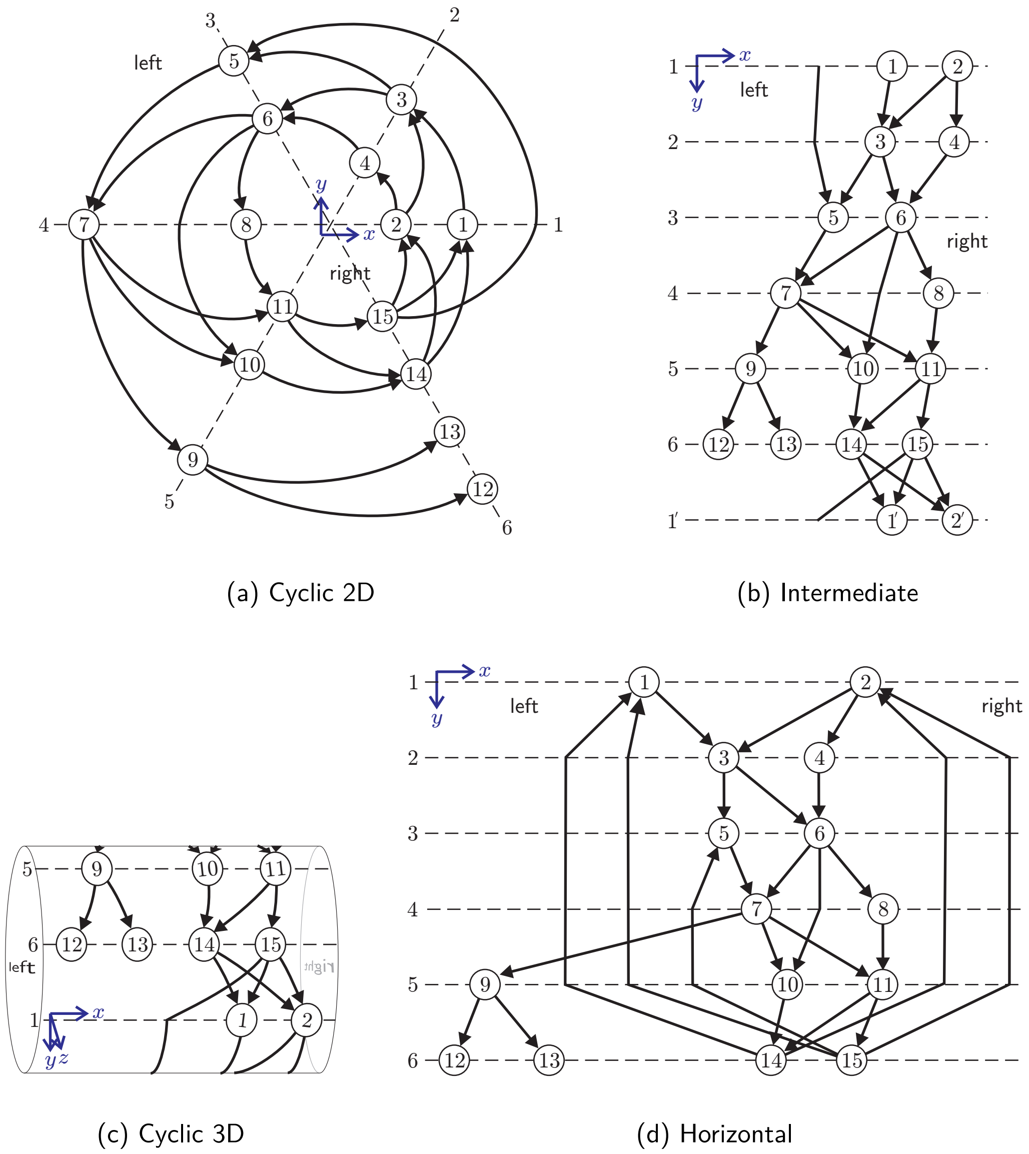


Figure 4.5. Up to four level pairs may enclose changes in crossings per swap



(a) Cyclic 2D

(b) Intermediate

(c) Cyclic 3D

(d) Horizontal

Figure 4.6. Different styles of cyclic drawings and directions

$r(p) = \Big(\delta' + \max\{x(v)|v \in V\}\Big) - x(p) \cdot \delta$ and the angle $\alpha(p) = \Big(y(p) - 1\Big) \cdot \frac{2\pi}{k}$. The constant value δ' defines the minimum distance of a vertex to the center as shift in x-direction and δ is as previously the constant minimum distance between vertices on the same level.

$$\Big(x_{\text{2D}}(p), y_{\text{2D}}(p)\Big) = \Big(r(p) \cdot \cos\Big(\alpha(p)\Big), r(p) \cdot \sin\Big(\alpha(p)\Big)\Big) \in \mathbb{R}^2 \tag{4.6}$$

A *cyclic 3D drawing* on a cylinder as exemplified by Fig. 4.6(c) uses for each point p the coordinates defined by (4.7), where r_k is the radius of the cylinder.

$$\Big(x_{\text{3D}}(p), y_{\text{3D}}(p), z_{\text{3D}}(p)\Big) = \Big(x(p) \cdot \delta, -r_k \cdot \sin\Big(\alpha(p)\Big), r_k \cdot \cos\Big(\alpha(p)\Big)\Big) \in \mathbb{R}^3 \tag{4.7}$$

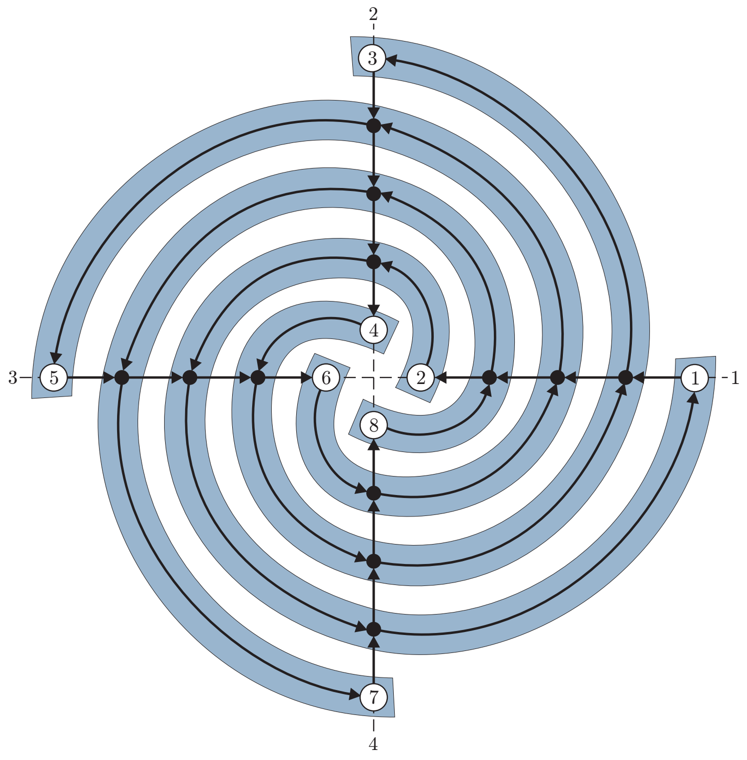
These equations transform straight-lined segments of the intermediate layout to spiral segments in the 2D or 3D drawings.

4.4.1 Layout Algorithm

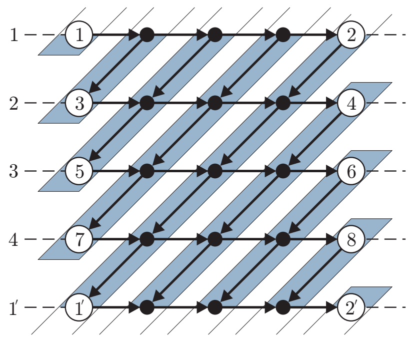
Similar to Sect. 1.2.4, the algorithm consists of three basic steps: block building, horizontal compaction, and balancing. The first two steps are carried out by four runs for each combination of left/right with up/down alignment. The results are merged by the balancing step. Again, we describe the left top run only, since the remaining runs can be realized by flipping the graph horizontally and/or vertically.

The block building phase is done in the same way as in [24]: We try to align vertices with its left upper median adjacent vertices to blocks and remove all other segments level by level until we obtain a cyclic block graph. The only difference is that the cyclic block graph can have *closed blocks* spanning exactly k levels as circles and *open blocks* spanning at least k levels as spirals which shall both be avoided to simplify the algorithm. In both cases we split such a block by removing outer segments until each resulting block has at most a span of $k - 1$. Suitable outer segments always exist, as no edge can span more than k levels. Thus, the invariant of at most two bends per edge still holds.

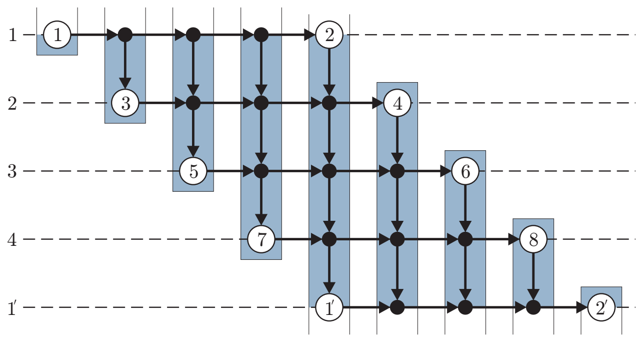
In the compaction step we compact the cyclic block graph by arranging all blocks as close as possible to each other minimizing the width of the drawing. The remaining discrepancy to [24] is that in the cyclic case there may be unavoidable cyclic dependencies in the left-to-right ordering among vertically aligned paths/blocks in the block graph, cf. Fig. 4.7(a). To see that, consider all intra-block edges undirected and find a cycle. Thus, in general it impossible to draw inner segments vertically (and straight-line), as shown by Fig. 4.7(c). We solve this problem by shearing the drawing of such cycles such that all inner segments have the same slope as done in Fig. 4.7(b). The slope has to be chosen such that each cycle and thus the resulting curve starts and ends at the same coordinates. For cyclic 2D drawings this means the edges are no circular arcs but "parallel" spirals with uniform radius growth factors. Since each block will be drawn with constant slope, the shearing has no influence to the guarantee of at most two bends per



(a) Cyclic



(b) Intermediate with sheared blocks



(c) Intermediate with vertical blocks (non-drawable since 1 and $1'$ resp. 2 and $2'$ are identical vertices with different coordinates)

Figure 4.7. Cyclic dependencies in block graph

edge. An originally closed block will not be sheared like other blocks as it cannot be part of a cyclic dependency.

In general there may be more than one "independent" cycle within the block graph. Hence, to obtain a drawing it is necessary to find an individual slope for each strongly connected component (SCC). Thus, we shear each SCC individually and all edges of each block in the same SCC get the same slope. The necessary and sufficient slope of an SCC S depends on the maximum number $\omega(S)$ of inter-block edges which can be found within one of its simple cycles. This corresponds to finding the longest path within a graph, which is in general $\mathcal{NP}$-hard [75]. Fortunately, in our special case it is possible to find this *widest cycle* C in linear time in the size of S, i. e., in $\mathcal{O}(|V(S)| + |\mathsf{E(S)}|)$ [3]. Let c_{down} and c_{up} be the numbers of intra-block edges traversed in C in and against their direction, respectively. The *number of windings* of S is then defined as $\mathrm{wind}(S) := \lfloor \frac{c_{\mathsf{down}} - c_{\mathsf{up}}}{k} \rfloor$. Informally speaking, $\mathrm{wind}(S)$ counts how often S wraps around the center. Choosing then an uniform slope of $\frac{-(\mathrm{wind}(S) \cdot k)}{\omega(S)}$ draws each inter-block edge of C with unit length. Other cycles in S which are not as wide as C will have some unused horizontal space in the drawing and, thus, will have intra-block edges longer than unit distance.

Finding C and compacting S is done simultaneously: We find a y-monotone path P in S from a rightmost vertex through an arbitrary *starting block* B to a leftmost vertex, using inter-block edges in both directions and preserving intra-block edge directions. By virtually removing all inter-block edges of P we cut S open. This cuts each cycle of the block graph exactly once. We assign an arbitrary vertex $v \in V$ of B the new coordinate $y'(v) = \phi(v)$. In a traversal of the block graph we then assign each other vertex a y'-coordinate: Using an inter-block edge we assign both end vertices the same y'-coordinate. Using an intra-block edge in or against its direction we increase or decrease the y'-coordinate by 1, respectively, without using a modulo operation. The result is an acyclic block graph, which we compact (in contrast to [24]) in the following way: After placing each block which is a source on an imaginary zero line, we compact S by a leftmost topological sorting preserving unit distance. Afterwards we fix all sinks on their positions, traverse all other blocks in reversed topological order, and move them as much to the right as possible, again preserving unit distance. For placing a block as close as possible to the already placed ones, we traverse its levels. After the compaction each block (and therefore each vertex v in S) has an assigned x'-coordinate. Let $e = (u, v)$ be a removed inter-block edge. The width of a simple widest cycle through e is then $x'(u) - x'(v) + 1$. Considering all removed inter-block edges and computing the maximum value gives the width of the widest simple cycle which defines the width $\omega(S)$ of S. Applying the modulo k operation on the y-coordinates gives the final intermediate layout of S.

After compacting each SCC on its own, we globally compact the set of SCCs. Since all vertices of each SCC on the same level are consecutive, no SCCs can interleave. We interpret the SCCs as super vertices and perform a topological sorting on the resulting DAG. We then compact the SCCs in the same way as we have compacted the blocks within one SCC before.

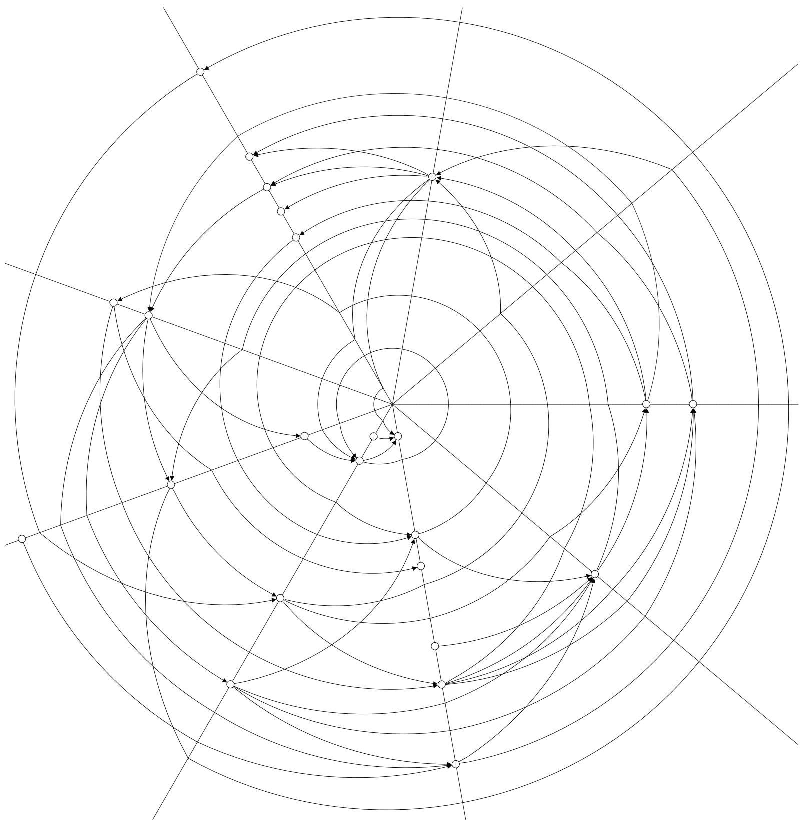
In the final balancing step the four results are combined for computing one x-coordinate for each vertex out of the four x-coordinates from the four runs. Contrary to [24], we cannot use the average median of the four x-coordinates for each vertex, since this may induce additional bends on edges in the cyclic case. The reason is that on lines with potentially different slopes the median changes at crossings, i. e., it is a non-linear function. Hence, we instead use the barycenter of all four x-coordinates for each vertex, which preserves the orderings of the vertices on a level. The result is a non-integral x-coordinate. However, due to the sheared drawing of the SCCs none of the four intermediate drawings has guaranteed integer x-coordinates, anyway. Additional bends can occur since the blocks of the four runs may differ. However, the invariant of at most two bends per edge e in the final drawing still holds, as the y-coordinates of the bends located at the topmost and lowest dummy vertex of e are identical in each drawing. It is possible that some vertices in one run belong to a block of an SCC although they do not belong to an SCC or even one block in another run. Thus, balancing can lead to more different slopes than in each of the four runs alone.

If similar slopes should be more important than balancing, we only perform one downwards run with a modified block building step. We start with the median vertex of the upper level and align it with its median successor. Whenever one of the two medians is not unique, we choose an arbitrary one. From this vertex we move to the left (right) to align the remaining vertices on the level trying the right (left) median first. However, this results in balanced outgoing edges only. We compact unsymmetrically to the left first as described previously.

For an embedding of a proper graph $G = (V, E, \phi)$ we obtain an intermediate drawing which has a width of $\mathcal{O}(\frac{|V|^2}{k})$ and an used area of $\mathcal{O}(|V|^2)$. For the 3D drawing the same bounds hold. The 2D drawing has a width and a height of $\mathcal{O}(\frac{|V|^2}{k})$ and thus an area of $\mathcal{O}(\frac{|V|^4}{k^2})$. Our layout algorithm has a linear time complexity $\mathcal{O}(|V| + |E|)$ in the size of the proper graph $G = (V, E, \phi)$.

4.4.2 Example Outputs

In the following, we present some example outputs of our cyclic framework generated with the JAVA implementation of [67] which should help to judge the quality of the resulting drawings.


(a) 2D drawing

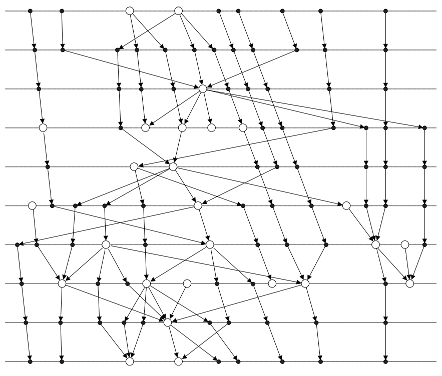
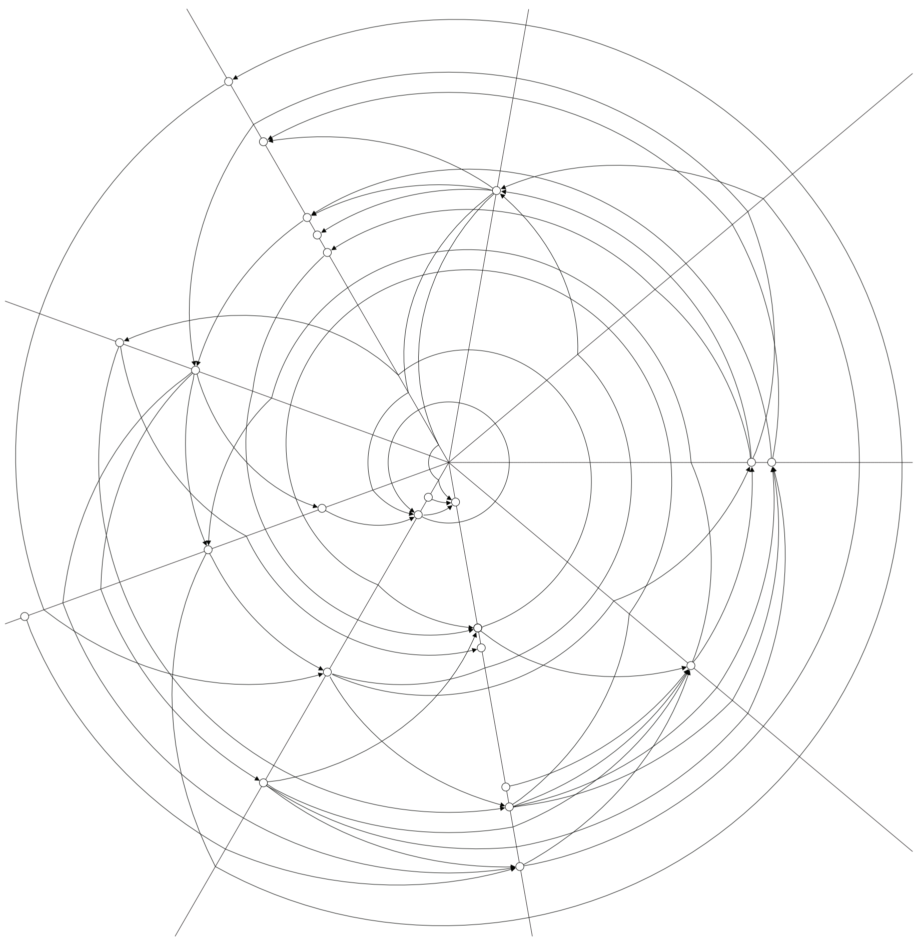

(b) Intermediate and surface of the 3D drawing

Figure 4.8. A graph with $|V| = 24, |E| = 44, k = 9$ after four runs


(a) 2D drawing

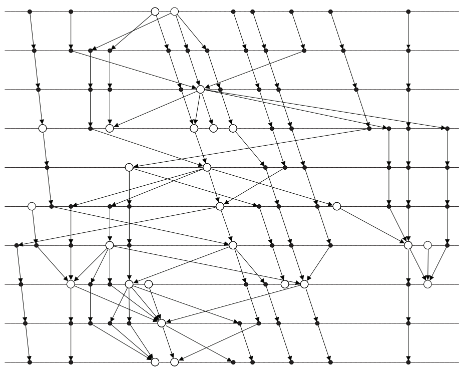

(b) Intermediate and surface of the 3D drawing

Figure 4.9. The graph of Fig. 4.8 after one balanced run

5 Conclusion

This thesis focuses on drawing directed graphs and presents a generalized framework for drawing graphs containing hierarchical or recurrent information. Most of the algorithms have been implemented in JAVA as plug-ins of Gravisto [8] or as standalone prototypes and have been evaluated theoretically and empirically.

5.1 Summary

After an introduction to well known topics in graph theory and graph drawing we gave an extended overview of the Sugiyama framework [138] with its four phases, cycle removal, level assignment, crossing reduction, and coordinate assignment. It is the most common and the de facto standard algorithm for drawing directed graphs. We then extended it in three directions: First we showed how to eliminate the level-by-level sweeping approach with its local view on crossings used for nearly thirty years now within most standard crossing reduction heuristics. We have introduced a practical global optimization considering in each step all affected crossings of the whole graph. With its elimination of type 2 conflicts, this global approach perfectly matches to the linear time layout algorithm of Brandes and Köpf [24]. This is up to now the best known algorithm for the coordinate assignment phase realizing crucial aesthetic criteria like vertical long edges and at most two bends per edge. We further showed how to additionally consider intra-level edges which often arise directly and/or in a natural way from the applications. This makes the algorithms more usable in practice.

Afterwards we showed enhancements phase by phase for concentric and star-shaped instead of parallel horizontal level lines. There are many similarities but there are also many discrepancies in the algorithmic problems and solutions addressed by the four phases. Some of the problems have an essentially different character. Particularly, for radial drawings the first two phases are nearly identical. For the third phase the algorithms have to be adapted to cope with two possible edge directions, i. e., clockwise or counter-clockwise routing. For the fourth phase an adequate cut has to be found, the traditional fourth phase applied, and then the resulting coordinates have to be transformed. For cyclic drawings, the first phase is not applicable. The second phase, i. e., the leveling, is completely different as in horizontal problems: A suitable number of levels k is not computed automatically by the algorithms. Further, the traditional algorithms cannot be applied since they always argue with source and sink vertices which in general do not exist in non-acyclic graphs. The $\mathcal{NP}$-hard optimization goals behind the algorithms are exactly opposite to each other. Crossing reduction can be done with the global crossing reduction without any major changes. The one-sided level-by-level approaches are not sufficient for that. The fourth phase has a different character, since the blocks must have some slope to obtain drawable layouts, i. e., vertical blocks are in general not possible.

From a software engineering perspective, all presented algorithms are modular and fit seamlessly together into one framework. The obtained time complexities of our proposed major approaches are summarized by Table 5.1.

Table 5.1. Time complexities of the proposed algorithms for a graph $G = (V, E)$ and (one of) its proper k-leveled version $G' = (V', E', \phi')$

	horizontal	radial	cyclic
cycle removal	$\mathcal{O}(\lvert E\rvert)$ [53]	$\mathcal{O}(\lvert E\rvert)$ [53]	–
leveling	$\mathcal{O}(\lvert V\rvert + \lvert E\rvert)$ [36]	$\mathcal{O}(\lvert V\rvert + \lvert E\rvert)$	$\mathcal{O}(\lvert V\rvert \log \lvert V\rvert$ $+k \cdot \deg(G) \cdot \lvert E\rvert)$
glob. cross. reduct.	$\mathcal{O}(\lvert E\rvert^2)$	$\mathcal{O}(\lvert E\rvert^3)$	$\mathcal{O}(\lvert E\rvert^2)$
coord. assignment	$\mathcal{O}(\lvert V'\rvert + \lvert E'\rvert)$ [24]	$\mathcal{O}(\lvert V'\rvert + \lvert E'\rvert)$	$\mathcal{O}(\lvert V'\rvert + \lvert E'\rvert)$

Clearly, the cylinder drawings in 3D may in general not be useful for an easy human perception, at least if visualized on a 2D display. However, using interactive and scrollable 2D views on the cylinders changes that. We even suggest to combine the best of both worlds radial and cyclic to obtain an interactive 2D view on the torus (genus 1 surface) which is infinitely scrollable in four directions. This avoids long edges striking out the whole drawing, but preserves the mental map of the classes and their interactions. See Fig. 5.1 for an example UML class diagram of the Java jar-API.

5.2 Future Work

Actually the phases of the hierarchical framework are not independent of each other. The modularization is to some degree only an algorithmical simplification. This is per se a good thing, but here the quality of the result can suffer severely. For example consider level assignment and crossing reduction. Each leveling potentially has an own

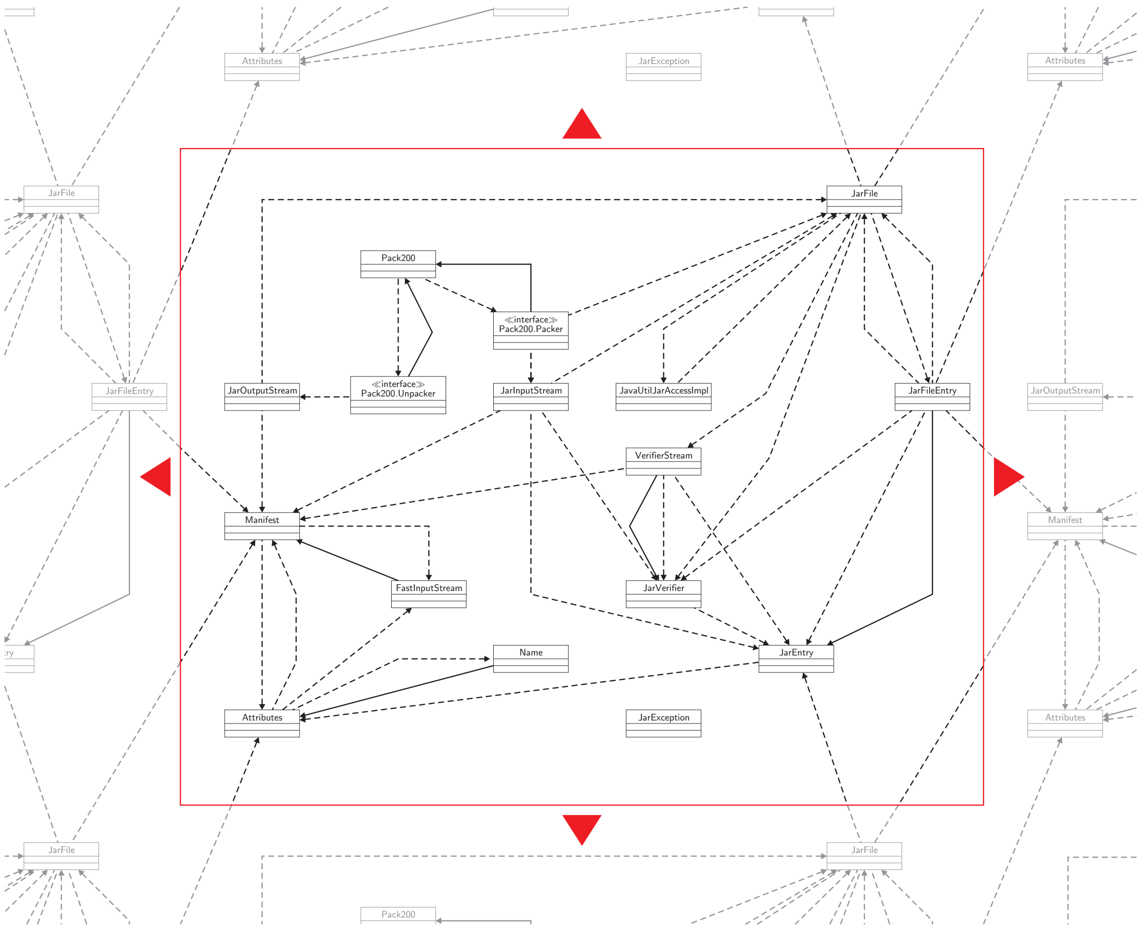


Figure 5.1. Interactive view of an UML class diagramm from the Java jar-API. Associations are drawn solid while ≪uses≫-connections are dashed

minimum number of crossings since vertices are only allowed to be permuted within their assigned level. Thus, for a low number of crossings an optimization is to consider the phases not independently of each other [33]. See [140] for such an approach using a genetic algorithm for combined level assignment and crossing reduction phases. Chimani et al. [33, 34] utilize upward planarity and planarization to introduce new levels for a low number of crossings. Our global crossing reduction approach is also a next step towards this goal, since it is especially conditioned to support coordinate assignment phases which forbid type 2 conflicts. Also, a longest path ordering of our blocks may serve as a basis for the coordinate assignment.

A completely different approach for crossing reduction is planarization [113, 114], see Sect. 1.3. In our eyes an important task would be to systematically extend planarization to radial and cyclic level graphs and for graphs with intra-level edges. The counterpart in standard horizontal drawings has been well studied [57, 115] and the resulting drawings seem to be pleasing [113, 114]. The results from the most recent work by Chimani et al. [33, 34] are very promising and achieve improvements for crossing reduction by up to 30% compared to a classical Sugiyama approach. Roughly speaking, they try to find an *upward drawing*, i. e., where all edges are pointing in one direction, of a DAG with as few crossings as possible. However, the price for this reduction is high. The convention of the hierarchic framework is lost as phase three introduces many new levels. The area is expanded and it is not clear whether there is one extra level for each served crossing. At present the achievements/improvements of their approach are incomparable to our approach. First heuristics for planarization in radial drawings are presented by Buchner [30].

These continuative questions and open problems will be addressed in an ongoing research paper supported by the German Research Foundation DFG: Is there any extension of speeding-up global sifting by the use of lower bounds similar to [77]? Note, a straight forward extension of [77] by using variables χ_{AB} which denote the number of crossings of edges incident to blocks A and B with $\pi(A) < \pi(B)$ instead of vertices is not possible. These crossings also depend on the positions of the other adjacent blocks which are variable. However, if there is any extension, does it also apply for the radial, cyclic, or intra-level edge cases? What tight lower and upper bounds do exist for the optimal crossing number χ over k-levels for the different kinds of drawing styles? How do they change if type 2 conflicts are forbidden? Are there any approximation ratios of the presented heuristics for crossing reductions, or even (complete) new approximation algorithms? Are there any efficient algorithms for counting existing crossings in cyclic and radial k-level embeddings as there are [15, 129, 142] for the horizontal case with two levels? Another goal is to extend global sifting to consider constraints, i. e., some given relative orders of blocks or vertices which may not be violated.

In this thesis we have shown algorithms for the first three phases which are independent of (up to $\mathcal{O}(k \cdot |E|)$) dummy vertices. This is reflected by their running times summarized in Table 5.1. This is not yet true for the fourth phases, which still need dummy vertices and thus have running times in $\mathcal{O}(|V'| + |E'|)$ instead $\mathcal{O}(|V| + |E|)$ for (embeddings of) graphs $G = (V, E, \phi)$ and their normalized versions $G' = (V', E', \phi)$. Eiglsperger et al. [59] presented an $\mathcal{O}((|V| + |E|) \log |E|)$ time horizontal coordinate assignment phase using the algorithm of Brandes and Köpf [24] with a minor modification. Remember, the original algorithm needs $\mathcal{O}(|V'|+|E'|)$ time. They mark type 1 conflicts already during their $\mathcal{O}((|V|+|E|)\log E)$ time crossing reduction phase by considering crossings of their *virtual edges* on each level. Then they use long inner segments within the Brandes/Köpf algorithm, i. e., they consider only up to two extremal dummy vertices per long edge. They claim, that there are no further changes. Marking type 1 conflicts as crossings of outer segments and blocks is also possible while swapping two blocks in our global sifting approach without affecting time complexities. However, in [59, 60] the algorithm is not completely described, e. g., it is unclear how to build up the block graph as defined in [24]. Originally each dummy vertex gets an inter-block edge to its predecessor or successor, respectively. For example, there may be blocks A whose set of spanned levels is a subset of the set of levels of non-extremal dummy vertices of predecessor/successor blocks B. Then the (up to two) inter-block edges between A and B are undefined since there are no end vertices in A on the respective levels. A solution may be to create inter-block edges between blocks as super vertices rather than between (dummy) vertices. However, it is left open if at all and how to do that concretely and whether this has some consequences to the theoretical running time. Important future research should also contain a translation of this or a similar approach for radial and cyclic drawing styles.

The remainder lists some further interesting challenges on higher levels of detail which are specific for the drawing styles and their phases.

Radial Crossing Reduction The problem of finding an optimal offset for a radial two-level graph with fixed orderings of the vertices of both levels is still open. More specifically: Is it $\mathcal{NP}$-hard to test whether a bipartite graph $G = (V_1 \dot\cup V_2, E)$ with given positions π of V_1 and V_2 admits edge offsets ψ such that the resulting number of crossings is a most K? However, there exists a 3-approximation algorithm for the problem [116].

Cyclic Leveling What is the optimum number of k of levels for cyclic leveling? Further, if representing the cycles as such is not that important as the sum of the edge length, then one could consider to reverse some edges in the cyclic case as well. A detailed study of a combined edge reversal/leveling algorithm would be of interest.

Cyclic Coordinate Assignment Open problems for the cyclic coordinate assignment are a symmetric compaction for the one run version and alternative approaches for more compact drawings, i. e., with linear width.

A Selected Articles

In this appendix you find selected publications which are the base of this cumulative thesis. The articles are ordered as the ideas and contents are summarized in Sections 2, 3, and 4.

A.1 Articles for Sect. 2

This section lists a preprint of [11] and [4] in this order.

Crossing Minimization in Extended Level Drawings of Graphs

Christian Bachmaier [a,*], Hedi Buchner [b], Michael Forster [a], Seok-Hee Hong [c]

[a] *University of Passau, Faculty of Inf. and Math., 94030 Passau, Germany.*

[b] *IMAGEN Program, National ICT Australia, Eveleigh, NSW 1430, Australia.*

[c] *School of Information Technologies, University of Sydney, NSW 2006, Australia.*

Abstract

The most popular method of drawing directed graphs is to place vertices on a set of horizontal or concentric levels, known as level drawings. Level drawings are well studied in Graph Drawing due to their strong application for the visualization of hierarchy in graphs. There are two drawing conventions: horizontal drawings use a set of parallel lines and radial drawings use a set of concentric circles.

In level drawings, edges are only allowed between vertices on different levels. However, many real world graphs exhibit hierarchies with edges between vertices on the same level. In this paper, we initiate the new problem of extended level drawings of graphs, which was addressed as one of the open problems in social network visualization, in particular, displaying centrality values of actors. More specifically, we study minimizing the number of edge crossings in extended level drawings of graphs. The main problem can be formulated as the extended one-sided crossing minimization problem between two adjacent levels, as it is folklore with the one-sided crossing minimization problem in horizontal drawings.

We first show that the extended one-sided crossing minimization problem is $\mathcal{NP}$-hard for both horizontal and radial drawings, and then present efficient heuristics for minimizing edge crossings in extended level drawings. Our extensive experimental results show that our new methods reduce up to 30% of edge crossings.

Key words: (radial) level graph, crossing minimization, intra-level edges, hierarchy, level/layered drawing, visualization of social networks, graph algorithm

* Corresponding author.
Email addresses: `bachmaier@fim.uni-passau.de` (Christian Bachmaier), `hedi.buchner@nicta.com.au` (Hedi Buchner), `forster@fim.uni-passau.de` (Michael Forster), `shhong@it.usyd.edu.au` (Seok-Hee Hong).

1 Introduction

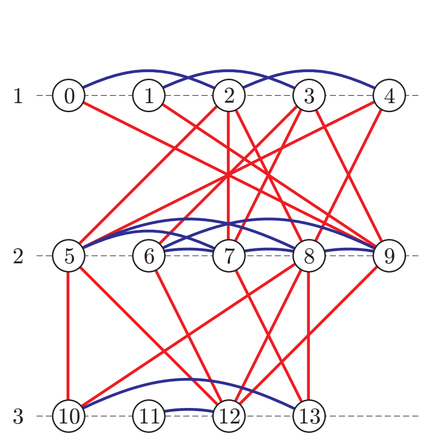
A *level drawing* (or *hierarchical drawing*) of a graph is the most popular drawing convention for directed graphs, alternatively known as the *Sugiyama method* [1]. Consequently, drawing level graphs is a well-studied problem in Graph Drawing. There is a rich literature on drawing level graphs including characterizations of level-planar graphs, level planarity testing, crossing minimization, and planarization methods for non-level planar graphs, see [2].

There are two drawing conventions for level graphs: in *horizontal drawings*, vertices are placed on parallel horizontal lines and edges are drawn as strictly y-monotone polylines that may bend when they intersect a level line [1–3]. In *radial drawings*, vertices are placed on concentric circles and edges are drawn as polyline segments of spirals which are monotone from the concentric center to the outside [4]. Both drawings are produced based on the same drawing framework, the Sugiyama method, which consists of the following four steps:

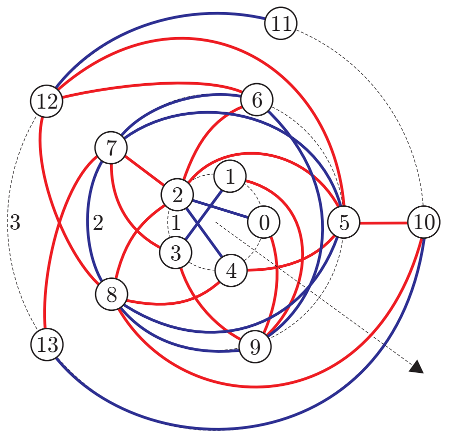
(1) *Cycle removal*: Reverse appropriate edges to eliminate cycles.
(2) *Level assignment*: Assign vertices to levels such that no edges have both end vertices on the same level, and introduce dummy vertices to represent long edges which span more than one level by a path of proper edges. The dummy vertices represent edge bends.
(3) *Crossing minimization*: Compute a good ordering of the vertices on each level to minimize edge crossings between two adjacent levels.
(4) *Coordinate assignment*: Assign x-/angular coordinates to the vertices to meet some esthetic criteria. The y-/radial coordinates are implicit through the levels.

However, many real world graphs exhibit hierarchies with edges between the vertices on the same level. For example, the visualization of *centrality* of *actors* in social networks produces level graphs with both *inter-level* and *intra-level* edges [5]. Note that up to now it is neither shown if intra-level edges reduce the visual complexity nor if they reduce the overall number of crossings. The fact that all existing hierarchical drawing methods more or less simply ignore intra-level edges, although they are present from the respective application in most cases, justifies an investigation. To our best knowledge, the only exception is the *compound graph* drawing algorithm of Sugiyama and Misue [6] where a fast but qualitatively inferior barycenter strategy on intra-level edges is used to avoid crossings of edges and bounding rectangles of a *compound node*.

We initiate the new problem of drawing *extended level graphs*, i. e., level graphs with intra-level edges. Drawing extended level graphs was addressed as one of the open problems in social network visualization by Brandes [7]. The proposed goal is an easy human perception, where one of the main criteria seems to be



(a) Horizontal.



(b) Radial.

Fig. 1. Example drawings.

an overall low number of crossings [8]. Extended level graphs often occur in practice, for example graphs where the level assignment is already predefined by breadth first search to express distances, or social networks where the importance (centrality) of an actor (modeled by a vertex) defines its level [9–11]. More specifically, the visualization of an actor's status in a social network is a horizontal drawing where the levels are not equidistant, because each level represents a real-valued centrality index for the actors [5]. Since the centrality values often differ only marginally, status values can be clustered. The actors with centrality values in the same range are assigned to the same level to avoid perceptual problems of having too many levels. However, this approach introduces many intra-level edges. In the conclusion of [5], Brandes et al. state that the treatment of intra-level edges needs further investigation. Later, Brandes proposed a new research direction on minimizing all types of crossings including inter-level edge and intra-level edges, for social network visualization as an open problem [7].

Another application of extended level graphs with radial drawings is the visualization of *micro/macro graphs* [12], e. g., arising from group analysis or role assignment in social networks [11]. In general, intra-level edges may help to gain better aspect ratios, since drawings tend to be much longer than wide, especially with the Sugiyama method.

In our *extended level drawings* of extended level graphs, we represent intra-level edges using circular arcs with different radii in order to avoid overlapping edges and crossings between vertices and edges, see Fig. 1. Further, we restrict to drawing the arcs only on one-side of the level lines, say above (inside), in order to model the problem as in the Sugiyama framework.

Table 1
Survey of treated edge crossing variants.

	inter-level	mixed	intra-level	all
horizontal drawings	[2]	✓	[13]/✓	✓
radial drawings	[4]	✓	✓	✓

In this paper, we study the crossing minimization problem in extended level graphs to improve the readability [8] of the extended level drawings. More precisely, we study the new problem of minimizing the number of edge crossings in extended level drawings of graphs. The main focus of this paper can be formally defined as the *extended one-sided crossing minimization* problem between two adjacent levels, similar to the well-known *one-sided crossing minimization* for the horizontal drawing convention. We show that the one-sided crossing minimization problem for extended level graphs is $\mathcal{NP}$-hard for both horizontal and radial drawings, and present greedy heuristics for minimizing edge crossings that take different types of edge crossings into account: *inter-level crossings* between two inter-level edges, *intra-level crossings* between two intra-level edges, and *mixed crossings* between intra-level edges and inter-level edges. Note that greedy heuristics are state of the art in this area [2] since they are much faster and, thus, can treat larger graphs than common local search algorithms.

Our main aim is to extend the well-known *sifting* heuristic for level drawings. More specifically, we designed new *extended sifting* heuristics by carefully integrating sifting, radial sifting, and circular sifting methods together with a new crossing counting algorithm. Our extensive experimental results show that our new methods reduce up to 30% of crossings compared to existing standard heuristics which only consider inter-level edge crossings. Of course, this value is only a rule of thumb for reasonable ratios between inter-level and intra-level edges. Since our algorithm is the first one which does not simply ignore intra-level edge crossings, it is clear that the more intra-level edges are present, the higher the gain will be. The running times of our algorithms are within the same bounds as the traditional sifting algorithms. Thus, we are able to handle the same graph sizes within similar times.

The checkmarks in Tab. 1 summarize the problems solved in this paper, which is organized as follows: After explaining some necessary preliminaries in Sect. 2, we present our extended sifting heuristics for extended level graphs, which explicitly consider three different types of crossings in horizontal drawings in Sect. 3. Then, we present extended radial sifting heuristics for extended level graphs in radial drawings in Sect. 4. Section 5 presents our experimental results and Sect. 6 concludes with some open problems.

2 Preliminaries

A (*proper*) *k-level graph* $G = (V, E, \phi)$ is a graph with a level assignment $\phi\colon V \to \{1, 2, \ldots, k\}$, which partitions the vertex set into $k \leq |V|$ pairwise disjoint subsets $V = V_1 \dot{\cup} V_2 \dot{\cup} \cdots \dot{\cup} V_k$, $V_i = \phi^{-1}(i)$, $1 \leq i \leq k$, such that $|\phi(u) - \phi(v)| = 1$ for each *inter-level edge* $\{u, v\} \in E$. Particularly, $k = 1$ implies that $E = \emptyset$. For $v \in V$ with $\phi(v) > 1$ let $E(v) = \{\, \{u, v\} \in E \mid u \in V_{\phi(v)-1} \,\}$ be the (predecessor) *inter-level adjacency list.* Define $E(v) = \emptyset$, if $\phi(v) = 1$. An *ordering* of a level graph is a partial order $\prec$ of V such that $u \prec v$ or $v \prec u$ iff $\phi(u) = \phi(v)$ for each pair of vertices $u, v \in V$. If the vertex sets V_i are ordered sets (according to $\prec$), we call G an *ordered level graph.*

2.1 Sifting with a Crossing Matrix

The most common technique for crossing minimization in level drawings is to only consider two consecutive levels at a time in multiple top-down and bottom-up passes. Starting with an arbitrary ordering of the first level, subsequently the ordering of one level is fixed, while the subsequent level is reordered to minimize the number of crossings in-between. Thus, the 2-level horizontal drawing is the fundamental building block for drawing level graphs with k-levels.

The well-studied *one-sided* 2*-level crossing minimization problem* is formally defined as follows: Given a 2-level graph $G = (V_1 \dot{\cup} V_2, E, \phi)$, where the vertex set V_1 is given with a fixed ordering, compute an ordering of V_2 which produces the minimum number of crossings. This is known to be $\mathcal{NP}$-hard [14] and a number of heuristics, approximation algorithms, and exact algorithms have been proposed. Eades and Wormald [14] proposed a *median heuristic*, which produces a 3-approximate solution to the one-sided crossing minimization problem. The *barycenter heuristic* by Sugiyama et al. [1] is an $O(\sqrt{n})$-approximation [14]. The barycenter (median) heuristic assigns each vertex of V_2 the barycenter (median) value of its neighbors in V_1, assuming the positions of vertices in V_1 are numbered from 1 to $|V_1|$ according to $\prec$. A sorting according to these values defines the ordering among the vertices in V_2. Currently, the best known approximation algorithm for the one-sided crossing minimization problem given by Nagamochi [15] delivers 1.4664-approximate solutions. Jünger et al. [16, 17] presented integer linear programming algorithms and experimentally compared the exact results with various heuristics. See [2, 3] for an extended overview.

For our new problem of one-sided crossing minimization in extended level graphs, we will adopt the sifting heuristic, which is slower than simple heuristics like barycenter or median heuristics, however, produces fewer crossings in practice. Sifting was originally introduced as a heuristic for vertex minimization in ordered binary decision diagrams [18] and later adapted for the one-sided crossing minimization problem [19]. The main idea is to keep track of the objective function while moving in a *sifting step* a vertex $u \in V_2$ along with a fixed ordering of all other vertices in V_2 and then placing u to its locally optimal position. This is done by iteratively swapping consecutive vertices only.

The method is thus an extension of the *greedy-switch* heuristic [20], where u is swapped iteratively with its successor. We call a single swap a *sifting swap*. Executing a sifting step for every vertex in V_2 is called a *sifting round*. For crossing minimization, the objective function is the number of crossings between the edges incident to the vertex under consideration and all other edges. The efficient computation of the crossing count in sifting is based on the *crossing matrix*. The $|V_2|^2$ entries in the crossing matrix correspond to the number of crossings caused by (the edges of) pairs of vertices in a particular relative ordering and can be computed as a preprocessing step in $\mathcal{O}(|E|^2)$ time [21, 22]. Whenever a vertex is placed in a new position, only a small number of updates is necessary. This allows a running time of $\mathcal{O}(|V_2|^2)$ for one round. In practice, only few sifting rounds ($3 - 5$ for reasonable problem instances) are necessary to reach a local optimum for all vertices simultaneously. Our experiments showed that this is in most cases also the global optimum which we computed for small graphs with the ILP formulation of [17]. The largest reduction of crossings usually occurs in the first round.

2.2 Crossing Minimization in Radial Drawings

Compared to the horizontal drawings of level graphs, radial drawings of level graphs have not been well studied. The problem of crossing minimization in radial drawing is more challenging, as it involves both vertex ordering and edge routing problems. That is, even if the orderings of vertices in both orbits are fixed, we still need to decide how to route (i. e. clockwise or counterclockwise) each edge around the inner orbit in order to minimize the number of edge crossings in a radial drawing.

Bachmaier [4] presented a new radial drawing framework, an adaptation of the Sugiyama method [1] to radial drawings. He proved that the one-sided crossing minimization problem in radial drawings is $\mathcal{NP}$-hard and presented a number of heuristics including *radial sifting* with experimental results. The first polynomial time 15-approximation algorithm for one-sided crossing minimization problem in radial drawings was presented by Hong and Nagamochi [23]. Their main contribution was to reduce a given instance of the one-sided crossing minimization in a radial drawing to that of the one-sided crossing minimization in a horizontal drawing.

As our new *extended radial sifting* heuristics for extended level graphs is based on radial sifting, we will explain details including basic terminologies in Sect. 4.

2.3 Circular Sifting

The asymptotic overall running time of the original algorithm described above is $\mathcal{O}(|E|^2 + |V_2|^2)$ and too high for our purposes, i. e., to handle large graphs. Thus, we apply the *circular sifting heuristic* of Baur and Brandes [13] used for the $\mathcal{NP}$-hard [24] crossing minimization problem in circular drawings: Order the vertices V of a graph $G = (V, E)$ which all are placed on a single circle, e. g., as in Fig. 5, to minimize the number of crossings among the straight-line edges in E. Since there is no "circular" order, Baur and Brandes define linear orders $\prec_\alpha$ by selecting a reference vertex $\alpha \in V$ which is the first of the (here counterclockwise) sequence. For finding the locally optimal position of a vertex $u \in V$ in a sifting step, it is sufficient to record the change in crossing count while swapping u with its successor $v_p \in V$. This can be done by considering only edges incident to u or v_p: After a swap exactly those pairs of these edges cross which did not cross before. All other crossings remain unchanged (let $\chi(\pi)$ be the number of crossings of a drawing π and $N(v)$ be the set of adjacent vertices of $v \in V$).

Lemma 2.1 (Baur, Brandes) *Let $u \prec_u v_p \in V$ be consecutive vertices in a circular drawing π and let π' be the drawing with their positions swapped, then*

$$\begin{aligned}\chi(\pi') = \chi(\pi) - &\sum_{x \in N(u)} |\{\, y \in N(v_p) \mid y \prec_x^{\pi} u \,\}| \\ &+ \sum_{y \in N(v_p)} |\{\, x \in N(u) \mid x \prec_y^{\pi'} v_p \,\}| \ .\end{aligned}$$

At the end of one step, u is placed where the intermediary crossing counts reached their minimum. For efficiency reasons, the computation of the change in crossing count is implemented over suffix lengths in ordered adjacency lists.

2.4 Inter-Level Sifting for Crossings between Inter-Level Edges

For horizontal level lines, we adapt the above idea to one-sided 2-level crossing minimization, which we call *inter-level sifting* for simplicity. We mainly exchange $\prec_\alpha$ by $\prec$ and virtually connect the start and the end points of the level lines to obtain a circle. Then, we only consider the ordering of the permutable level 2 as presented by Algorithms 1, 2, and 3. We obtain the same results as with the matrix method, without knowing the absolute crossing numbers, however. Since all three methods are generic and are also used for the following algorithms, Algorithm 2 already contains lines 4 and 8. At present, these lines can be ignored and the input graphs can be considered as $G = (V_1 \dot\cup V_2, E, \phi)$ for ease of understanding. For efficiency reasons, all shown operations are implemented in place on the graph data structure.

Algorithm 1. SIFTING-ROUND

```
Input: Ordered 2-level graph $G = (V_1 \dot\cup V_2, E, H, \phi)$
Output: Updated ordering of $V_2$
1  foreach $u \in V_2$ do
2      $V_2 \leftarrow$ SIFTING-STEP($G$, $u$)
3  return $V_2$
```

Algorithm 2. SIFTING-STEP

```
Input: Ordered 2-level graph $G=(V_1 \dot\cup V_2, E, H, \phi)$, Vertex $u \in V_2$ to sift
Output: Updated ordering of $V_2$
1  let $v_0 = u \prec v_1 \prec \cdots \prec v_{|V_2|-1}$ be the current ordering of $V_2$ with $u$ put to front
2  foreach $v \in V_2$ do
3      Sort $E(v) \subseteq E$ on ascending ordering of $V_1$ in $\mathcal{O}(|E|)$ time
4      Sort $H_l(v), H_r(v) \subseteq H$ on ascending ordering of $V_2$ in $\mathcal{O}(|H|)$ time
5  $\chi \leftarrow 0$; $\chi^* \leftarrow 0$            // current and best number of crossings
6  $p^* \leftarrow 0$                     // best vertex position
7  for $p \leftarrow 1$ to $|V_2| - 1$ do
8      $l \leftarrow$ UPDATE-INTRA-ADJ($G$, $u$, $v_p$)
9      $\chi \leftarrow \chi$+SIFTING-SWAP-INTER($G$, $u$, $v_p$)
10     if $\chi < \chi^*$ then
11         $\chi^* \leftarrow \chi$; $p^* \leftarrow p$
12 return $V_2 \leftarrow v_1 \prec \cdots \prec v_{p^*-1} \prec u \prec v_{p^*} \prec \cdots \prec v_{|V_2|-1}$
```

3 Crossing Minimization on Horizontal Levels

An *extended k-level graph* $G = (V, E, H, \phi)$ is a k-level graph (V, E, ϕ) which additionally has *intra-level edges* $\{u, v\} \in H$ with $\phi(u) = \phi(v)$. For $v \in V_i$ let $H_l(v) = \{\, \{u, v\} \in H \mid u \prec v \,\}$ be the *left intra-level adjacency list* and $H_r(v) = \{\, \{v, w\} \in H \mid v \prec w \,\}$ be the *right intra-level adjacency list.*

In this section, we first consider the new problem of one-sided 2-level crossing minimization for extended level graphs with horizontal drawings. It is easy to see that the one-sided 2-level crossing minimization problem for an extended 2-level graph is $\mathcal{NP}$-hard, since at least two subproblems, considering only

Algorithm 3. SIFTING-SWAP-INTER

Input: Ordered 2-level graph $G = (V_1 \dot\cup V_2, E, H, \phi)$, Swap vertices $u, v_p \in V_2$
Output: Change in crossing count

```
1  let $x_0 \prec \cdots \prec x_{r-1}$ be the neighbors of $u$ in $V_1$
2  let $y_0 \prec \cdots \prec y_{s-1}$ be the neighbors of $v_p$ in $V_1$
3  $c \leftarrow 0$; $i \leftarrow 0$; $j \leftarrow 0$
4  while $i < r$ and $j < s$ do
5      if $x_i \prec y_j$ then
6          $c \leftarrow c + (s - j)$
7          $i \leftarrow i + 1$
8      else if $y_j \prec x_i$ then
9          $c \leftarrow c - (r - i)$
10         $j \leftarrow j + 1$
11     else
12         $c \leftarrow c + (s - j) - (r - i)$
13         $i \leftarrow i + 1$; $j \leftarrow j + 1$
14 return $c$
```

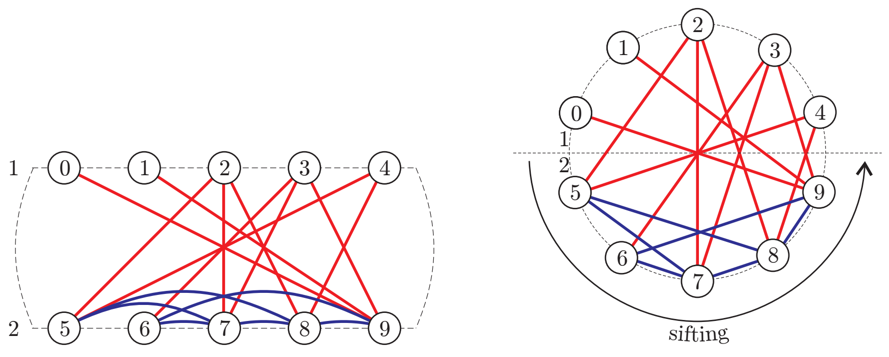
inter-level edges [14] and considering only intra-level edges [24] are $\mathcal{NP}$-hard. The circular crossing minimization in [24] is exactly the same as minimizing crossings among intra-level edges of a horizontal level i (consider the level line i bent to a circle).

Lemma 3.1 *The one-sided 2-level crossing minimization problem for extended level graphs in horizontal drawings is $\mathcal{NP}$-hard.*

This motivates us to design efficient heuristics for the problem and we design extensions of the sifting heuristic for extended level graphs in order to compute a reasonable solution efficiently. After presenting a simple integrated method that only works for horizontal drawings, we then present our main method, extended sifting heuristics for horizontal drawings, which consists of three subroutines, each minimizing different types of crossings.

3.1 Compact Method

The extended one-sided 2-level crossing minimization problem on horizontal levels can be transformed into a corresponding circular crossing minimization problem where the vertices of the two levels are placed on the two disjoint semicircles, i. e., vertices on level 1 clockwise on semicircle 1 and vertices on level 2 counterclockwise on semicircle 2, see Fig. 2. Then, only vertices in V_2 are sifted using positions in semicircle 2 only. This is a minor modification of the original circular sifting.



(a) 2-level graph. (b) Problem transformation.

Fig. 2. Using circular sifting for extended one-sided 2-level crossing minimization.

Definition 3.1 *For a 2-level graph $G = (V_1 \dot{\cup} V_2, E, H, \phi)$ let π be a horizontal drawing with*

$$u_1 \prec^\pi u_2 \prec^\pi \cdots \prec^\pi u_{r=|V_1|} \quad \text{and} \quad v_1 \prec^\pi v_2 \prec^\pi \cdots \prec^\pi v_{|V_2|}$$

for vertices $u_i, r \in V_1$, $1 \leq i \leq |V_1|$, and $v_j \in V_2$, $1 \leq j \leq |V_2|$.

A corresponding circular drawing π' *is a circular drawing of G with*

$$u_{|V_1|} \prec_r^{\pi'} u_{|V_1|-1} \prec_r^{\pi'} \cdots \prec_r^{\pi'} u_1 \prec_r^{\pi'} v_1 \prec_r^{\pi'} v_2 \prec_r^{\pi'} \cdots \prec_r^{\pi'} v_{|V_2|-1} \ .$$

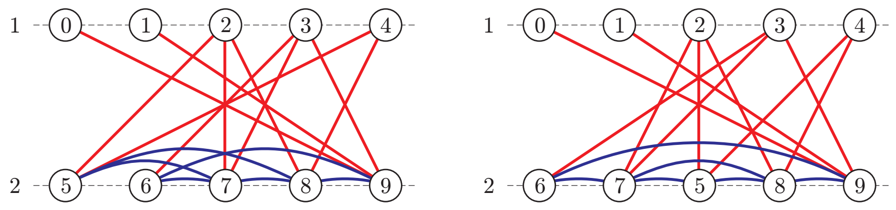
Lemma 3.2 *Let π' be a corresponding circular drawing of a 2-level graph $G = (V_1 \dot{\cup} V_2, E, H, \phi)$ drawing π. Then two edges cross in π' if and only if they cross in π.*

PROOF. Let $e_1 = (u_1, v_1)$, $e_2 = (u_2, v_2) \in E$ be two arbitrary inter-level edges with w. l. o. g. $u_1 \prec^\pi u_2$. They cross in π if and only if $v_2 \prec^\pi v_1$, whereas they cross in π' if and only if $u_2 \prec_{v_1}^{\pi'} u_1$ and $v_2 \prec_{u_1}^{\pi'} v_1$.

Now, let $e_1 = (v_1, v_3)$, $v_2 = (v_2, v_4) \in H$ be two arbitrary intra-level edges with w. l. o. g. $v_1 \prec^\pi v_2$. They cross in π if and only if $v_2 \prec^\pi v_3 \prec^\pi v_4$, whereas they cross in π' if and only if $v_1 \prec_{v_1}^{\pi'} v_2 \prec_{v_1}^{\pi'} v_3 \prec_{v_1}^{\pi'} v_4$.

Finally, let $e_1 = (u, v_2) \in E$ be an arbitrary inter-level edge and $e_2 = (v_1, v_3) \in H$ be an arbitrary intra-level edge with w. l. o. g. $v_1 \prec^\pi v_3$. They cross in π if and only if $v_1 \prec^\pi v_2 \prec^\pi v_3$, whereas they cross in π' if and only if $v_1 \prec_u^{\pi'} v_2 \prec_u^{\pi'} v_3$ and $v_3 \prec_{v_3}^{\pi'} u \prec_{v_3}^{\pi'} v_1$. □

Unfortunately, this simple transformation cannot be applied to radial drawings. Thus, we present a new method for extended one-sided crossing minimization in the following section.



(a) Before minimization. (b) After minimization.

Fig. 3. The first two levels of Fig. 1.

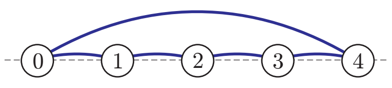
3.2 Extended Sifting

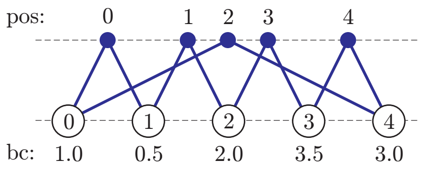
We now present a new *extended sifting* algorithm for extended one-sided crossing minimization, where we treat the three different kinds of crossings separately without any impact on the running time and quality. The algorithm runs not only theoretically but also practically within the same time and generates exactly the same orderings and number of crossings. Further, speaking in terms of algorithm or software engineering, the problem is attacked with a more modular approach.

3.2.1 Intra-Level Sifting for Crossings Between Intra-Level Edges

Consider overlapping intra-level edges $\{v_1, v_4\}, \{v_2, v_3\} \in H$ with $v_1 \prec v_2 \prec v_3 \prec v_4$. They do not cross, since we draw each edge $\{u, v\}$ as a circular arc instead of a straight line. For that, we use a quadratic spline with an amplitude, i. e., height of the only interpolation point, rising with the number of enclosed vertices between u and v in the current ordering $\prec$ of V_2. Thus, even if $v_1 = v_2$ or $v_3 = v_4$ the edges do not cross, except in common end points. We further take care not to introduce unnecessary "double" crossings between intra-level and inter-level edges by restricting the maximum edge amplitude according to the dimension of the drawing. For example, if the edge $\{5, 8\}$ in Fig. 3(a) has a higher amplitude, it would cross the edge $\{4, 5\}$. Note that we require to draw all intra-level edges completely above the second level line, as will be explained in Sect. 3.2.2.

Note that Sugiyama and Misue [6] presented a faster but qualitatively inferior *insertion barycenter method* for intra-level crossing minimization. More specifically, they created a dummy vertex splitting each intra-level edge, which they placed on a common dummy level. After computing the barycenter value of the neighbors for each dummy vertex, they ordered the dummy level in ascending value. Finally, they computed the barycenter values for the original vertices according to the new positions of their dummy neighbors, which define the final ordering. Unfortunately, this method introduces unnecessary crossings, as


(a) Before.



(b) Intermediate.

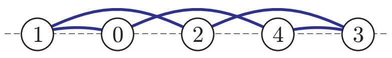

(c) After.

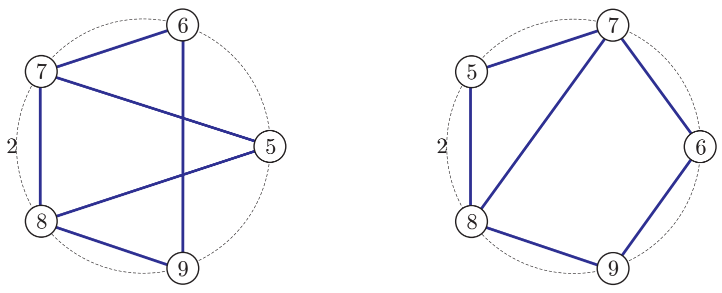
Fig. 4. Insertion Barycenter of [6].

Fig. 4 shows. A straightforward solution to avoid these unnecessary crossings may be to make the amplitudes of the edges pairwise different, which leads up to $|H|$ different dummy levels. As a consequence, in order to be able to run a 2-level crossing minimization algorithm considering all types of crossings later, each inter-level edge must be split in $|H| + 1$ segments by $|H|$ new dummy vertices. This prevents not only time efficient processing, but also is obstructive for a good result, i. e., fewer crossings: Each of the additionally necessary $|H|$ crossing minimization rounds is a heuristic only and is thus not exact.

Thus, we again use the idea from circular sifting (Sect. 2.3), which we already have used in inter-level sifting in Sect. 2.4, however, now for crossing minimization between intra-level edges. Hence, we call it *intra-level sifting*.

Considering the horizontal line of level 2 bent to a circle (see Fig. 5), the circular crossing minimization algorithm fits out of the box: For one round call Algorithm 1 where line 9 of Algorithm 2 is changed to call Algorithm 4 instead of Algorithm 3. Line 3 of Algorithm 2 is left away in this case. Algorithm 4 is the same as Algorithm 3 except that the neighbors are on level 2 and the ordering $\prec$ is replaced by $\prec_{v_p}$, i. e., the ordering of V_2 is different in each swap.

With Algorithm 5 we keep the ordered intra-level adjacencies of vertex u up to date during a sifting step. Thus, we know the ordering $\prec_{v_p}$ among u's neighbors, since this is the concatenation of $H_r(u)$ and $H_l(u)$ (in this order). Therefore, we need no reordering for determining the x_is per swap. The same holds for the y_is: Algorithm 5 also updates the intra-level adjacencies of the swap vertex v_p, but does not maintain their ordering due to performance restrictions, in contrast to u. However, we rely on the fact that a *short edge* $h = \{u, v_p\}$ is always the first of $H_l(v_p)$. This is true since we build up the sorting of this adjacency list right after u was placed on the first position of V_2 in Algorithm 2 and there never were any updates to this ordering. In



(a) Before. (b) After.

Fig. 5. Circular crossing minimization for intra-level edges of the graph in Fig. 3.

Algorithm 4. SIFTING-SWAP-INTRA

Input: Ordered 2-level graph $G = (V_1 \dot\cup V_2, E, H, \phi)$, Swap vertices $u, v_p \in V_2$
Output: Change in crossing count

```
let $x_0 \prec_{v_p} \cdots \prec_{v_p} x_{r-1}$ be the neighbors of $u$ in $V_2 - \{v_p\}$, $\{x_i, u\} \in H$
let $y_0 \prec_{v_p} \cdots \prec_{v_p} y_{s-1}$ be the neighbors of $v_p$ in $V_2 - \{u\}$, $\{y_j, v_p\} \in H$
$c \leftarrow 0$; $i \leftarrow 0$; $j \leftarrow 0$
while $i < r$ and $j < s$ do
    if $x_i \prec_{v_p} y_j$ then
        $c \leftarrow c - (s - j)$
        $i \leftarrow i + 1$
    else if $y_j \prec_{v_p} x_i$ then
        $c \leftarrow c + (r - i)$
        $j \leftarrow j + 1$
    else
        $c \leftarrow c - (s - j) + (r - i)$
        $i \leftarrow i + 1$; $j \leftarrow j + 1$
return $c$
```

other words, the ordering of the intra-level adjacencies of all vertices v_p is valid throughout the complete sifting step besides obsolete positions of edges $\{u, v_p\}$. However, these exceptions are irrelevant for the determination of the orderings of the y_is, since they never contain u.

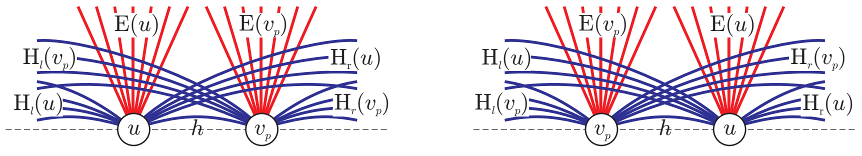
3.2.2 Mixed Sifting for Crossings Between Inter-Level and Intra-Level Edges

As mentioned previously, we restrict intra-level edges to be only routed above the second level line. Otherwise, if we allowed routing on both sides, the number of crossings between inter-level and intra-level edges would depend on the inter-level edges to vertices on a third level, which contradicts the pairwise level by level sweep approach.

Algorithm 5. UPDATE-INTRA-ADJ

Input: Ordered 2-level graph $G=(V_1 \dot\cup V_2, E, H, \phi)$, Swap vertices $u, v_p \in V_2$
Output: Number of edges between u and v_p, Updated $H_l(u)$, $H_r(u)$, $H_l(v_p)$, and $H_r(v_p)$ as side effect

```
1 $l \leftarrow 0$                                   // number of short intra-level edges
2 while $\{u, v_p\} = getFirst(H_r(u)) \in H$ do
3     $h \leftarrow$ removeFirst($H_r(u)$)
4     append($H_l(u)$, $h$)
5     removeFirst($H_l(v_p)$)                      // first, since list was never updated before
6     prepend($H_r(v_p)$, $h$)
7     $l \leftarrow l + 1$
8 return $l$
```



(a) Before swap. (b) After swap.

Fig. 6. Crossings among intra-level and inter-level edges.

As with the previous inter-level and intra-level sifting algorithms, swapping vertex u with its successor v_p changes only crossings (here among inter-level and intra-level edges) between edges incident to u or v_p. Thus for computing the change in the crossing count, we only need the sizes of the six sets $H_l(v)$, $H_r(v)$, $E(v)$ with $v \in \{u, v_p\}$, see Fig. 6.

Neglecting potentially existing short edges $h = \{u, v_p\} \in H$ which is a non-contributing special case, we obtain (1) as change in crossing count Δ when swapping u and successor v_p. The correctness follows again from the invariant that after a swap exactly those pairs of intra-level (excluding short edges) and inter-level edges cross which did not cross previously.

$$\Delta = (|H_r(v_p)| - |H_l(v_p)|) \cdot |E(u)| + (|H_l(u)| - |H_r(u)|) \cdot |E(v_p)| \qquad (1)$$

Thus, for *mixed sifting* a complete round can be started by calling Algorithm 1 and updating line 9 of Algorithm 2 to call Algorithm 6. Line 3 of Algorithm 2 does not need to be executed here. Note that the intra-level adjacency updates caused by a swap are done prior to a call of Algorithm 6. Thus l has now to be subtracted from $H_l(u)$ and $H_r(v_p)$ instead of $H_r(u)$ and $H_l(v_p)$.

Algorithm 6. SIFTING-SWAP-MIXED

```
Input: Ordered 2-level graph $G=(V_1 \dot\cup V_2, E, H, \phi)$, Swap vertices $u, v_p \in V_2$,
       Number of short edges $l = |\{\{u, v_p\} \in H\}|$
Output: Change in crossing count
1 return $((|H_r(v_p)| - l) - |H_l(v_p)|) \cdot |E(u)| + ((|H_l(u)| - l) - |H_r(u)|) \cdot |E(v_p)|$
```

3.3 Combining all Crossings

Finally, for our main algorithm *extended sifting* considering all types of crossings, we call Algorithm 1 with an updated line 9 of Algorithm 2 in order to call Algorithm 7. There, we simply add the three independent changes in crossing counts. However, other formulas preferring some type of crossings at the expense of more crossings of other types are possible, e. g., the usage of weighting factors.

Algorithm 7. SIFTING-SWAP-EXT

```
Input: Ordered 2-level graph $G=(V_1 \dot\cup V_2, E, H, \phi)$, Swap vertices $u, v_p \in V_2$
Output: Change in crossing count
1 $c_E \leftarrow$ SIFTING-SWAP-INTER($G$, $u$, $v_p$)
2 $c_H \leftarrow$ SIFTING-SWAP-INTRA($G$, $u$, $v_p$)
3 $c_{HE} \leftarrow$ SIFTING-SWAP-MIXED($G$, $u$, $v_p$)
4 return $c_V + c_H + c_{HV}$
```

We obtain the same time bound as the original sifting algorithm for a level graph $G = (V, E, \phi)$ considering only inter-level edges, or for a graph $G = (V, H)$ considering only intra-level edges.

Theorem 3.1 *One round of extended one-sided sifting on an extended* 2*-level graph* $G = (V, E, H, \phi)$ *takes* $\mathcal{O}(|V| \cdot (|E| + |H|))$ *time.*

PROOF. For running time calculations, we assume w. l. o. g. that there are no isolated vertices. They can be removed in preprocessing step and added again in postprocessing since their positions have no influence on the crossing number.

One round of inter-level (intra-level) sifting takes $\mathcal{O}(|V| \cdot |E|)$ ($\mathcal{O}(|V| \cdot |H|)$) time according to Theorem 3 of [13]. One round of mixed sifting takes $\mathcal{O}(|V| \cdot |H|)$ time, since one step needs $\mathcal{O}(|H|)$ time: The initial sorting of the intra-level adjacency in Algorithm 2 can be done in $\mathcal{O}(|H|)$ time by traversing the vertices of V_2 in order and adding each to the adjacency lists of its right or left neighbors. Each of the $|V_2|$ sifting swaps takes constant time. An integrated execution is possible, since the only updates to the intra-level adjacency list

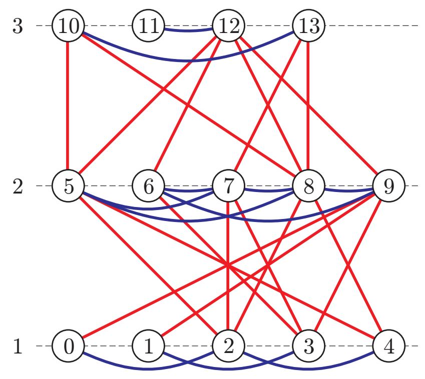


Fig. 7. Bottom-up sweep for the graph in Fig. 1.

are done by Algorithm 2. Thus the algorithms mutually do not compromise each other. □

3.4 Sweep over all Levels

According to our experience, the quality of sifting does not depend much on the quality of the initial vertex ordering of the first level. However, a "bad" initialization raises the number of needed sifting rounds and thus the absolute running time. Therefore, it may be useful to apply some rounds of intra-level sifting to V_1 to get a practical initial ordering.

In a top-down sweep, we reorder the levels i from 2 to k by consecutively applying our extended one-sided 2-level crossing minimization on the fix ordered set V_{i-1} and on the freely permutable set V_i. In the subsequent bottom-up sweep we reorder the levels i from $k-1$ down to 1 by consecutively applying the extended one-sided 2-level crossing minimization on the fix ordered set V_{i+1} and the permutable set V_i. However, in the bottom-up sweep we have a slightly different situation, since the intra-level edges are below the current level i and cross edges from level i and $i-1$ (see Fig. 7 for an example). Nevertheless, the formula for crossings of intra-level and inter-level edges (1) does not depend on any vertex ordering different to that on level i and especially does not depend on that of level $i-1$. Thus, we count the change in the number of crossings of the intra-level edges of level i with the inter-level edges between level i and $i-1$ during a swap. For this, we let for every $v \in V_i$ be $E(v) = \{\,(x, v) \mid x \in V_{i-1}\,\}$ instead of $\{\,(y, v) \mid y \in V_{i+1}\,\}$ in Algorithm 6, which then is the same as a top-down sweep. After some iterations (for our experiments 10) of top-down with subsequent bottom-up sweeps the algorithm terminates.

4 Crossing Minimization on Concentric Levels

In this section, we present a heuristic for extended one-sided crossing minimization with *radial drawings*. In radial drawings, we place vertices on concentric circles, instead of parallel horizontal lines [4,25]. The major advantage of radial over horizontal drawings is the additional freedom of routing edges in two directions around the center, i. e., clockwise and counterclockwise, which results in fewer edge crossings and therefore reduced visual complexity. Further, there is also a higher probability of no crossings, because of the fact that the set of level planar graphs is a proper subset of radial planar graphs [25].

As in the previous section, to display extended k-level graphs $G = (V, E, H, \phi)$, we define *extended radial drawings* as radial drawings with additional intra-level edges. For crossing minimization in extended radial drawings, we use exactly the same framework as for horizontal drawings, described in the previous section. Thus, we restrict ourselves to the only difference, the *one-sided radial* 2*-level crossing minimization*, in the following.

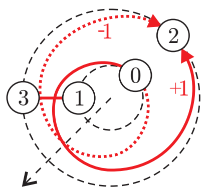
4.1 Radial Sifting for Crossing Minimization in Radial Drawings

In this section we briefly review the results and terminology of *radial sifting* [4] for crossing minimization in radial drawings of level graphs $G = (V_1 \dot\cup V_2, E, \phi)$ containing inter-level edges, an adaption of the original sifting algorithm to radial drawings.

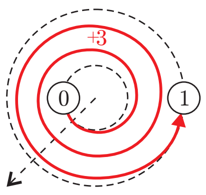
In order to represent orderings π_1 and π_2 (w. l. o. g. counterclockwise) of the vertices on the circular levels, a *ray* is introduced as a straight half-line from the concentric center to infinity which tags the borderline between the vertices with extremal positions. Edges crossing the ray are called *cut edges.*

How many times and in which *direction* an edge is wound around the center is crucial for radial drawings. This information is stored by the *offset* $\psi\colon E \to \mathbb{Z}$ of an edge. Thereby, $|\psi(e)|$ counts the crossings of an edge $e \in E$ with the ray. If $\psi(e) < 0$ ($\psi(e) > 0$), e is a *clockwise* (*counterclockwise*) cut edge, i. e., the sign of $\psi(e)$ reflects the mathematical direction of rotation, see Fig. 8. If $\psi(e) = 0$, then e is no cut edge and thus needs no direction information. Observe that a cut edge cannot cross the ray clockwise and counterclockwise simultaneously and for a small number of crossings only offsets in $\{-1, 0, 1\}$ are of interest. A *radial embedding* $\mathcal{E}$ of G is defined by the vertex order π and the edge offsets ψ, i. e., $\mathcal{E} = (\pi, \psi)$.

Lemma 4.1 ([4]) *Radial one-sided* 2*-level crossing minimization is* $\mathcal{NP}$*-hard.*



(a) Edge $(0, 2)$ drawn counter-clockwise and clockwise (dotted).



(b) $\psi((0, 1)) = +3$.

Fig. 8. Offsets of edges [4].

For radial drawings, however, the idea of circular sifting cannot be adopted directly, as the crossing number between inter-level edges also depends on their offsets, which are not necessarily constant. Thus inter-level sifting as described in Sect. 2.3 is used, with a different formula for counting crossings ($\mathrm{sgn} : \mathbb{R} \to \{-1, 0, 1\}$ is the signum function):

Lemma 4.2 ([4]) *Let $\mathcal{E} = (\pi, \psi)$ be a radial embedding of a 2-level graph $G = (V_1 \dot{\cup} V_2, E, \phi)$. Then the number of crossings between two edges $e_1 = (u_1, v_1) \in E$ and $e_2 = (u_2, v_2) \in E$ is*

$$\chi_{\mathcal{E}}(e_1, e_2) = \max\left\{0, \left|\psi(e_2) - \psi(e_1) + \tfrac{b-a}{2}\right| + \tfrac{|a|+|b|}{2} - 1\right\},$$

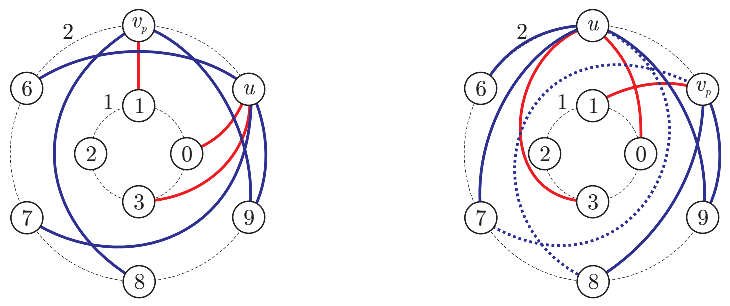
where $a = \mathrm{sgn}\,(\pi_1(u_2) - \pi_1(u_1))$ and $b = \mathrm{sgn}\,(\pi_2(v_2) - \pi_2(v_1))$.

Before counting the change in crossing number by considering the edges incident to those two swapped vertices, it is crucial to adapt the offsets of the edges. Let u be the vertex moved along in counterclockwise direction in a sifting step. Initially, the offset of all edges $(\cdot, u)$ are set to 1 and ordered according to the positions of incident vertices on level 1. While moving u along level 2 in counterclockwise direction, the offsets of edges are decreased according to their ordering by 1 as long as that reduces the number of crossings. The split in the edge list is called the *parting*, i. e., where the offsets differ by 1. The parting may move around the center twice, as offsets can be decreased from 1 to -1.

Theorem 4.1 ([4]) *Given a 2-level graph $G = (V, E, \phi)$, radial sifting runs in $\mathcal{O}(|V|^2 \cdot |E|)$ time.*

4.2 Intra-Level Edges in Extended Radial Drawings

We now discuss the new problem of *extended radial crossing minimization* for extended 2-level graphs $G = (V_1 \dot{\cup} V_2, E, H, \phi)$. For an *extended radial drawing* we draw intra-level edges as segments of circles with different radii (with the



(a) Compute number of crossings before swap.

(b) Update number of crossings after swap.

Fig. 9. Extended radial sifting: crossings between inter-level and intra-level edges and optimal routing.

interpolation point in the middle) according to the edge lengths, i. e., the number of spanned vertices plus 1. The interpolation point of longer edges is closer to the concentric center, see Fig. 1(b) for an example. Intra-level edges lie in the inner face of level 2, but must not intersect with the circle of level 1. Further, they must be "flat" enough, i. e., they must nestle to the circle of level 2 close enough, not intersecting inter-level edges of their end vertices unnecessarily. Contrary to straight-line edges, there are two possibilities to wind the edges around the center, clockwise or counterclockwise. For a low crossing number and low visual complexity, we always use the direction with shorter length, i. e., $\leq \lfloor \frac{|V_2|}{2} \rfloor$. See Fig. 9(b) where *long edges* are marked by dotted lines. For an easy notation $(v_1, v_2) \in H$ denotes an intra-level edge that is wound counterclockwise around the center starting at vertex v_1 and ending at vertex v_2. This partitions the intra-level adjacency $H(v)$ for each $v \in V_2$ in two sets, the *incoming set* $H_l(v) = \{ h = (\cdot, v) \mid h \in H \}$ and the *outgoing set* $H_r(v) = \{ h = (v, \cdot) \mid h \in H \}$. Per convention, we store edges for which each direction results in the same length in H_l. H_l and H_r are again kept sorted similar to Algorithm 5. As a consequence, both the left and the right adjacency lists of both vertices are updated after their sifting swap.

4.3 Extended Radial Sifting for Crossing Minimization in Extended level Graphs

In this section, we present our *extended radial sifting* heuristic for the extended one-sided radial crossing minimization problem. We first analyze the time complexity of the problem.

Lemma 4.3 *Extended radial one-sided* 2*-level crossing minimization is* $\mathcal{NP}$*-hard.*

PROOF. It is straightforward that the extended one-sided radial crossing minimization is $\mathcal{NP}$-hard, as the two subproblems, circular crossing minimization [24] and the radial one-sided 2-level crossing minimization [4] are $\mathcal{NP}$-hard. □

For extended radial sifting we also use the same modular approach as in extended intra-level sifting described in Sect. 3.2 to reduce the three different types of crossings, and add the resulting changes using the separate crossing numbers. For the minimization of crossings only between inter-level edges, the radial sifting heuristic is used as it is. Also, drawing intra-level edges as segments of circles instead of straight-lines clearly does not change the number of crossings between them. Thus, we can use the circular sifting heuristic. However, the algorithm for crossing minimization of crossings between intra-level edges and inter-level edges needs some modification, as will be described in the following section.

4.3.1 Crossings Between Inter-Level and Intra-Level Edges in Extended Radial Drawings

Since we have split the intra-level adjacencies in incoming and outgoing edges, computing the change in crossings when swapping two consecutive vertices u and v_p stays principally the same as in (1). Thereby, we again neglect short edges $\{u, v_p\}$ which do not contribute to the crossings. What remains is the additional freedom of routing the intra-level edges around the center in two different directions. Contrary to crossings between only intra-level edges, this now has an effect, see Fig. 9 for an example. To overcome this problem, we use the heuristic to always prefer the shorter direction.

We denote intra-level edges that span at least $\lfloor \frac{|V_2|}{2} \rfloor - 1$ vertices as *long edges*. After the swap, the length of all incoming intra-level edges of u, in $H_l(u)$ and all outgoing intra-level edges of v_p in $H_r(v_p)$ is increased by 1. Likewise, the length of all outgoing edges of u in $H_r(u)$ and all incoming edges of v_p in $H_l(v_p)$ is decreased by 1. In the case of an increase, it only can happen that the first (last) edge of the adjacency list $H_l(u)$ ($H_r(v_p)$) becomes a long edge. This is true, since we keep the adjacency lists ordered according to $\prec_u$ ($\prec_{v_p}$) and ascending edge lengths. The necessary updates are done with Algorithm 8. The number of crossings with inter-level edges COUNT-MIXED-CROSSINGS(h) caused by an intra-level edge $h = (v_1, v_2)$ is the number of inter-level edges which have exactly one incident vertex between v_1 and v_2 and the other one outside.

One special case remains: If $|V_2|$ is even, then some intra-level edges may have the same length $\frac{|V_2|}{2}$ in both directions. We call them *vis-à-vis edges*, since

Algorithm 8. SHORTEN-LONG-EDGES

Input: Ordered 2-level graph $G=(V_1 \dot\cup V_2, E, H, \phi)$, Swapped vertices $v_p, u \in V_2$
Output: Change in crossing count

```
1  $c \leftarrow 0$
2  while $h = (o, u) = getFirst(H_l(u)) \in H$ is long do
3      $c \leftarrow c -$ COUNT-MIXED-CROSSINGS($h$)
4      removeFirst($H_l(u)$)
5      remove($H_r(o), h$)
6      $h \leftarrow (u, o)$                // swap direction
7      append($H_r(u)$, $h$)
8      prepend($H_l(o)$, $h$)
9      $c \leftarrow c +$ COUNT-MIXED-CROSSINGS($h$)
10 while $h = (v_k, o) = getLast(H_r(v_p)) \in H$ is long do
11     $c \leftarrow c -$ COUNT-MIXED-CROSSINGS($h$)
12     removeLast($H_r(v_p)$)
13     remove($H_l(o), h$)
14     $h \leftarrow (o, v_k)$              // swap direction
15     prepend($H_l(v_p)$, $h$)
16     append($H_r(o)$, $h$)
17     $c \leftarrow c +$ COUNT-MIXED-CROSSINGS($h$)
18 return $c$
```

they are incident to two vertices that are placed opposite to each other. In order to locally minimize the number of crossings, we break ties in favor of the direction that causes less mixed crossings as shown in Algorithm 9. We update the adjacency lists according to which direction of the current edge causes less crossings.

4.3.2 All Crossings in Extended Radial Drawings

The overall sifting swap is essentially analogous to Algorithm 7. First we compute the change in the number of crossings between inter-level edges, then between intra-level edges, and finally between intra-level and inter-level edges. However, the sifting step (Algorithm 10) is extended as more updating parts are needed: After the preliminary steps and swapping the current vertex with its successor in the sifting swap, the offsets and parting of the involved inter-level edges must be updated as well as the routing of some intra-level edges, depending on their length.

Algorithm 9. SHORTEN-VISAVIS-EDGES

Input: Ordered 2-level graph $G=(V_1 \dot\cup V_2, E, H, \phi)$, Swapped vertices $v_p, u \in V_2$
Output: Change in crossing count

```
$c \leftarrow 0$
while $h = (o,u) = getFirst(H_l(u)) \in H$ is vis-à-vis do
    $c_1 \leftarrow$ COUNT-MIXED-CROSSINGS($h$)
    removeFirst($H_l(u)$)
    remove($H_r(o), h$)
    $h \leftarrow (u,o)$                    // swap direction
    append($H_r(u)$, $h$)
    prepend($H_l(o)$, $h$)
    $c_2 \leftarrow$ COUNT-MIXED-CROSSINGS($h$)
    if $c_2 \leq c_1$ then $c \leftarrow c + c_2 - c_1$
    else undo changes
while $h = (v_k, o) = getLast(H_r(v_p)) \in H$ is vis-à-vis do
    $c_1 \leftarrow$ COUNT-MIXED-CROSSINGS($h$)
    removeLast($H_r(v_p)$)
    remove($H_l(o), h$)
    $h \leftarrow (o, v_k)$                 // swap direction
    prepend($H_l(v_p)$, $h$)
    append($H_r(o)$, $h$)
    $c_2 \leftarrow$ COUNT-MIXED-CROSSINGS($h$)
    if $c_2 \leq c_1$ then $c \leftarrow c + c_2 - c_1$
    else undo changes
return $c$
```

4.3.3 Computational Complexity

For our extended radial sifting, we use the radial sifting algorithm as a subcomponent. Thereby, we consider the numbers of intra-level and mixed crossings additionally to the inter-level crossings. We obtain the following time complexity:

Theorem 4.2 *One round of extended radial sifting on an extended 2-level graph $G = (V, E, H, \phi)$ takes $\mathcal{O}(|V|^2 \cdot (|E| + |H|))$ time.*

PROOF. The worst case in terms of computational complexity occurs for the current vertex u being a vertex with inter-level and intra-level edges. In that case, the running time for one round of radial sifting is $\mathcal{O}(|V_2|^2 \cdot |E| + |E|^2)$ and for circular sifting $\mathcal{O}(|V_2| \cdot |H|)$.

Before and after a sifting swap, the computation and update of the mixed crossing number between two consecutive vertices u and v_p without any intra-level edges involved that change their direction runs in $\mathcal{O}(1)$ time. As in one

Algorithm 10. SIFTING-STEP

Input: Ordered 2-level offset graph $G=(V_1 \dot\cup V_2, E, H, \phi_E)$, Vertex $u \in V_2$ to sift
Output: Updated ordering of V_2

```
1  let $v_0 = u \prec v_1 \prec \cdots \prec v_{|V_2|-1}$ be the current ordering of $V_2$ with $u$ put to front
2  sort all edges
3  find best parting and offsets for edges $(\cdot, u) \in E$
4  $\chi \leftarrow 0$; $\chi^* \leftarrow 0$              // current and best number of crossings
5  $p^* \leftarrow 0$                                      // best vertex position
6  $j \leftarrow 0$; $j^* \leftarrow 0$                    // current and best offset at the parting
7  $i \leftarrow 0$; $i^* \leftarrow 0$                    // current and best parting
8  for $p \leftarrow 1$ to $|V_2| - 1$ do
9      $\chi \leftarrow \chi$+SIFTING-SWAP-EXT($G$, $u$, $v_p$)
10     $\chi \leftarrow \chi$+SHORTEN-LONG-EDGES($G$, $u$, $v_p$)
11     $\chi \leftarrow \chi$+UPDATE-OFFSETS($G$, $u$, $v_p$)
12     $\chi \leftarrow \chi$+SHORTEN-VISAVIS-EDGES($G$, $u$, $v_p$)
13     if $\chi < \chi^*$ then
14         $\chi^* \leftarrow \chi$; $p^* \leftarrow p$
15         $j^* \leftarrow j$
16         $i^* \leftarrow i$
17 set best parting and offsets for edges $(\cdot, u) \in E$
18 return $V_2 \leftarrow v_1 \prec \cdots \prec v_{p^*-1} \prec u \prec v_{p^*} \prec \cdots \prec v_{|V_2|-1}$
```

round of sifting each vertex is at each position once, this contributes $\mathcal{O}(|V_2|^2)$ to the overall running time. If an intra-level edge $h \in H$ changes direction, the computation of its number of crossings with inter-level edges runs in $\mathcal{O}(|E|)$ time. Now consider only one sifting step, i. e., one vertex u is moved along the periphery of its level. Let $h = (v_1, v_2)$ be an edge incident to two vertices $v_1, v_2 \in V_2$ with $v_1, v_2 \neq u$. Both v_1 and v_2 only change their position once during the sifting step of u. Thus, h can change its direction at most twice in one sifting step. Considering the incident intra-level edges of u, they can change their direction at most twice as well. Therefore, the contribution to the time complexity for one sifting step is $\mathcal{O}(|V_2| \cdot |E| + |H|)$ and thus for one round of sifting is $\mathcal{O}(|V_2|^2 \cdot |E|)$. □

5 Experimental Results

To analyze the performance of one sifting round of our extended one-sided 2-level crossing minimization heuristics, we have implemented them in Java. Further, we have implemented the corresponding standard sifting algorithm which uses a crossing matrix to compare its practical running time with the sifting algorithm of [13]. We have evaluated the implementations for horizontal drawings using 15625 random level graphs: 25 graphs for each combination of the parameters $|V_1| = |V_2| \in \{50, 100, \ldots, 1250\}$, $|E|/|V_2| \in \{1, \ldots, 5\}$, and $|H|/|V_2| \in \{1, \ldots, 5\}$. With the same parameters, but $|V_1| = |V_2| \in \{20, 30, \ldots, 290\}$, we similarly have tested 17500 random radial level graphs.

Figure A.2 and A.3 (Fig. A.11 and A.12 for concentric levels) confirm that it makes sense to consider all types of crossings simultaneously, since the algorithms generate (as expected) fewer crossings than standard sifting, experimentally by a factor of 0.7. This is a very encouraging result, since the differences in absolute running times between our extended sifting and the existing standard (inter-level) sifting and intra-level sifting, i. e., the running time of mixed sifting, are negligible in practice even on larger graphs (see Fig. A.1 and A.10). For example, in our experiment the running time of extended sifting for horizontal drawings with $|V_1| = |V_1| = |E| = |H| = 10^4$ is about 4 minutes and for radial drawings with $|V_1| = |V_1| = |E| = |H| = 10^3$ is about 6 minutes.

To give a feeling about the performance of inter-level sifting in horizontal drawings compared to an optimal algorithm: The ILP approach of [17] using the free lpsolve library needs for $|V_1| = |V_2| = 150$ and $|E| = 750$ about 50 minutes to reduce the number of crossings from 145925 to the optimum of 94742. One round of sifting needs less than 20 ms and leaves 94981 crossings. After 3 rounds we have 94770 crossings.

6 Conclusion

In this paper, we studied the new problem of crossing minimization in extended level graphs. We considered two different drawings, horizontal drawing and radial drawings, and presented two heuristics for extended one-sided crossing minimization in both drawings. Essentially, we extended the well-known sifting heuristic for crossing minimization of level graphs to handle three different types of crossings in extended level graphs. Ignoring non-contributing self loops, our algorithms can work also on multi-graphs within the same time bounds.

So far, we have used only random initial orderings of the vertices. However, the quality of the orderings produced by extended sifting is not independent from the input. Thus, it may be helpful to use some extensions of fast and simple heuristics, e. g., barycenter or median [2, 4] heuristics, to reduce crossings in a preprocessing step.

One future research is the investigation on the freedom of routing intra-level edges above and below the level lines, not restricting them to one side. As an alternative for the crossing minimization approach, a *planarization* approach was also studied for extended level graphs [26]: The planarization problem of extended-level graph is $\mathcal{NP}$-hard and, thus, heuristics are suggested. However, these should be evaluated with extensive experimental results.

References

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

[2] M. Kaufmann, D. Wagner, Drawing Graphs, Vol. 2025 of LNCS, Springer, 2001.

[3] G. Di Battista, P. Eades, R. Tamassia, I. G. Tollis, Graph Drawing: Algorithms for the visualization of Graphs, Prentice Hall, 1998.

[4] C. Bachmaier, A radial adaption of the sugiyama framework for visualizing hierarchical information, IEEE Transactions on Visualization and Computer Graphics 13 (3) (2007) 583–594.

[5] U. Brandes, J. Raab, D. Wagner, Exploratory network visualization: Simultaneous display of actor status and connections, Journal of Social Structure 2 (4) (2001) 1–28.

[6] K. Sugiyama, K. Misue, Visualization of structural information, IEEE Transactions on Systems, Man, and Cybernetics 21 (4) (1991) 876–891.

[7] S. Borgatti, S. Kobourov, O. Kohlbacher, P. Mutzel, Graph drawing with applications to bioinformatics and social sciences, Schloss Dagstuhl seminar 08191 (Mai 2008).

[8] H. C. Purchase, Which aesthetic has the greatest effect on human understanding?, in: G. Di Battista (Ed.), Proc. Graph Drawing, GD 1997, Vol. 1353 of LNCS, Springer, 1997, pp. 248–261.

[9] U. Brandes, P. Kenis, D. Wagner, Centrality in policy network drawings, in: J. Kratochvíl (Ed.), Proc. Graph Drawing, GD 1999, Vol. 1731 of LNCS, Springer, 1999, pp. 250–258.

[10] U. Brandes, P. Kenis, D. Wagner, Communicating centrality in policy network drawings, IEEE Transactions on Visualization and Computer Graphics 9 (2) (2003) 241–253.

[11] U. Brandes, T. Erlebach (Eds.), Network Analysis, Methodological Foundations, Vol. 3418 of LNCS Tutorial, Springer, 2005.

[12] M. Baur, U. Brandes, Multi-circular layout of micro/macro graphs, in: S. Hong, T. Nishizeki, W. Quan (Eds.), Proc. Graph Drawing, GD 2007, Vol. 4875 of LNCS, Springer, 2008, pp. 255–267.

[13] M. Baur, U. Brandes, Crossing reduction in circular layout, in: J. Hromkovic, M. Nagl, B. Westfechtel (Eds.), Proc. Workshop on Graph-Theoretic Concepts in Computer Science, WG 2004, Vol. 3353 of LNCS, Springer, 2005, pp. 332–343.

[14] P. Eades, N. C. Wormald, Edge crossings in drawings of bipartite graphs, Algorithmica 11 (1) (1994) 379–403.

[15] H. Nagamochi, An improved bound on the one-sided minimum crossing number in two-layer drawings, Discrete & Computional Geometry 33 (2005) 569–591.

[16] M. Jünger, E. K. Lee, P. Mutzel, T. Odenthal, A polyhedral approach to the multi-layer crossing minimization problem, in: G. Di Battista (Ed.), GD 1997, Vol. 1353, Springer, 1997, pp. 13–24.

[17] M. Jünger, P. Mutzel, 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms, Journal of Graph Algorithms and Applications 1 (1) (1997) 1–25.

[18] R. Rudell, Dynamic variable ordering for ordered binary decision diagrams, in: Proc. IEEE/ACM International Conference on Computer Aided Design, ICCAD 1993, IEEE Comp. Society Press, 1993, pp. 42–47.

[19] C. Matuszewski, R. Schönfeld, P. Molitor, Using sifting for k-layer straightline crossing minimization, in: J. Kratochvíl (Ed.), Proc. Graph Drawing, GD 1999, Vol. 1731 of LNCS, Springer, 1999, pp. 217–224.

[20] P. Eades, D. Kelly, Heuristics for reducing crossings in 2-layered networks, Ars Combinatorica 21 (A) (1986) 89–98.

[21] V. Valls, R. Martí, P. Lino, A branch and bound algorithm for minimizing the number of crossing arcs in bipartite graphs, Journal of Operational Research 90 (1996) 303–319.

[22] A. Yamaguchi, A. Sugimoto, An approximation algorithm for the two-layered graph drawing problem, in: T. Asano, H. Imai, D. T. Lee, S.-i. Nakano, T. Tokuyama (Eds.), Proc. International Conference on Computing and Combinatorics, COCOON 1999, Vol. 1627 of LNCS, Springer, 1999, pp. 81–91.

[23] S.-H. Hong, H. Nagamochi, Approximating crossing minimization in radial layouts, in: E. Sany Laber, C. F. Bornstein, L. T. Nogueira, L. Faria (Eds.), Proc. Latin American Theoretical Informatics Symposium, LATIN 2008, Vol. 4957 of LNCS, Springer, 2008, pp. 461–472.

[24] S. Masuda, T. Kashiwabara, K. Nakajima, T. Fujisawa, On the $\mathcal{NP}$-completeness of a computer network layout problem, in: Proc. IEEE International Symposium on Circuits and Systems, 1987, pp. 292–295.

[25] C. Bachmaier, F. J. Brandenburg, M. Forster, Radial level-planarity testing and embedding in linear time, Journal of Graph Algorithms and Applications 9 (1) (2005) 53–97.

[26] H. Buchner, Displaying centrality of a network using orbital layout, Master's thesis, University of Passau/National ICT Sydney (2006).

A Benchmark Results

The appendix shows the practical performance of the algorithms. All benchmarks were run on a 2.4 GHz Core 2 PC under the Java 6.0 platform from Sun Microsystems, Inc.

A.1 Horizontal Level Lines

Figures A.1 to A.9 provide benchmark results comparing the heuristics to minimize crossings in horizontal drawings: inter-level sifting (CM with crossing matrix as in Sect. 2.1, ES without as in Sect. 2.2), circular sifting with semi-circles (CS) as in Sect. 3.1, intra-level sifting (HS) as in Sect. 3.2.1, mixed sifting (MS) as in Sect. 3.2.2, and extended sifting (XS) as in Sect. 3.3.

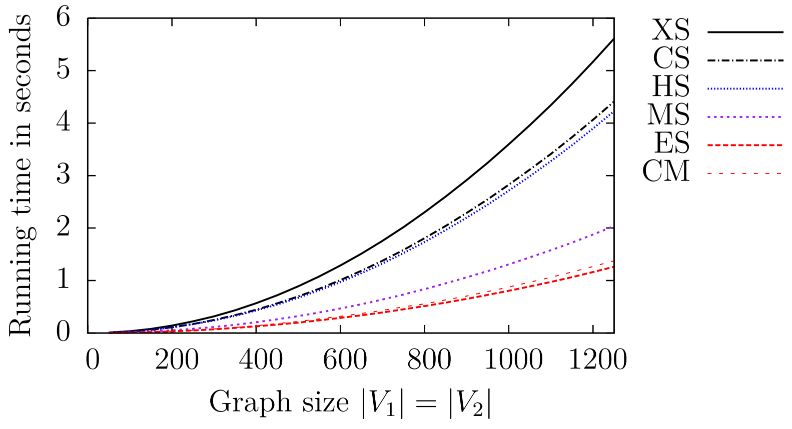


Fig. A.1. Benchmark: running times.

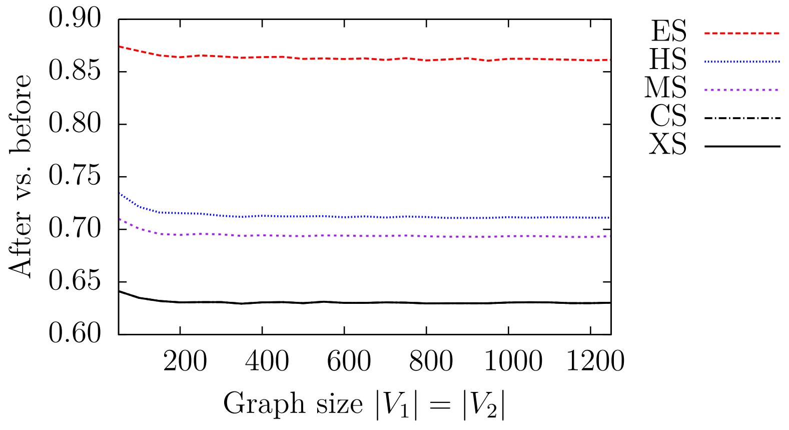


Fig. A.2. Benchmark: total crossing numbers.

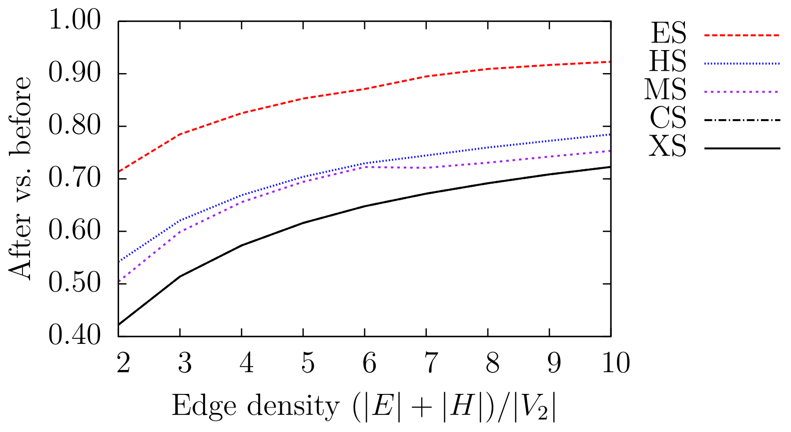


Fig. A.3. Benchmark: total crossing numbers.

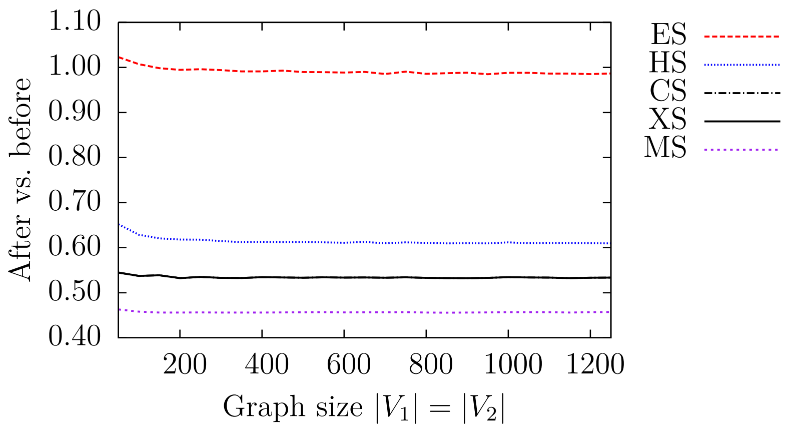


Fig. A.4. Benchmark: numbers of crossings between intra-level and inter-level edges.

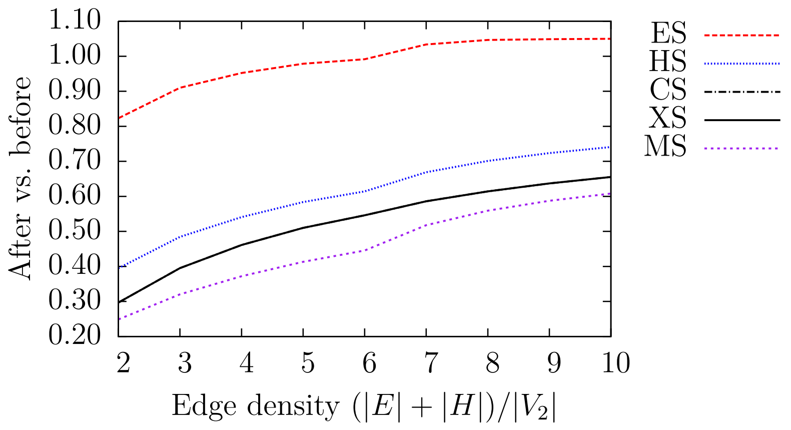


Fig. A.5. Benchmark: numbers of crossings between intra-level and inter-level edges.

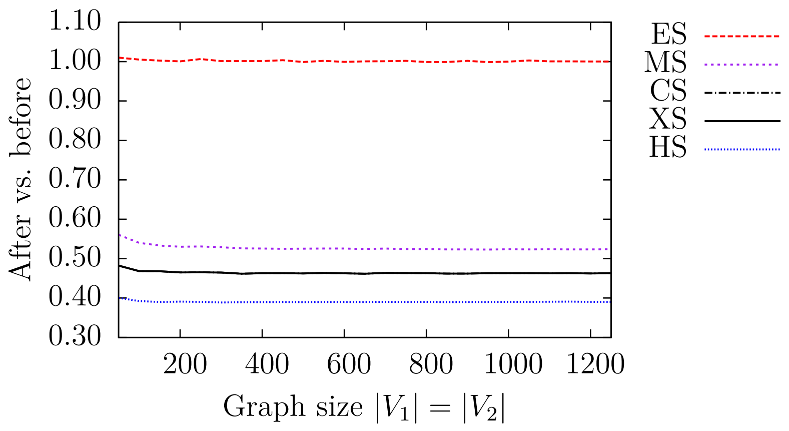


Fig. A.6. Benchmark: numbers of crossings between intra-level edges.

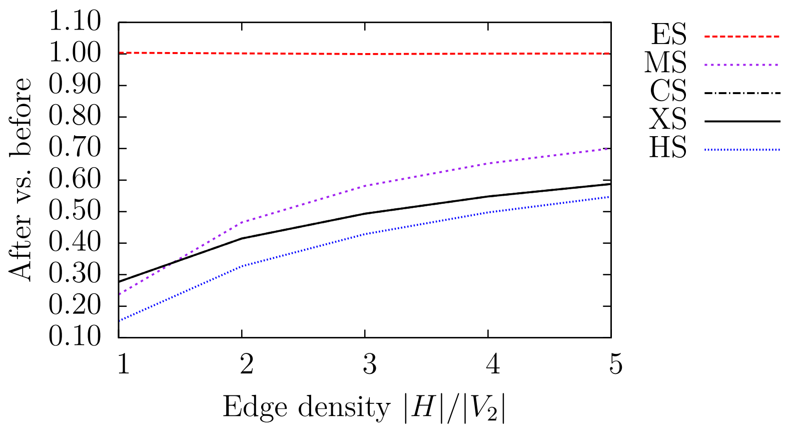


Fig. A.7. Benchmark: numbers of crossings between intra-level edges.

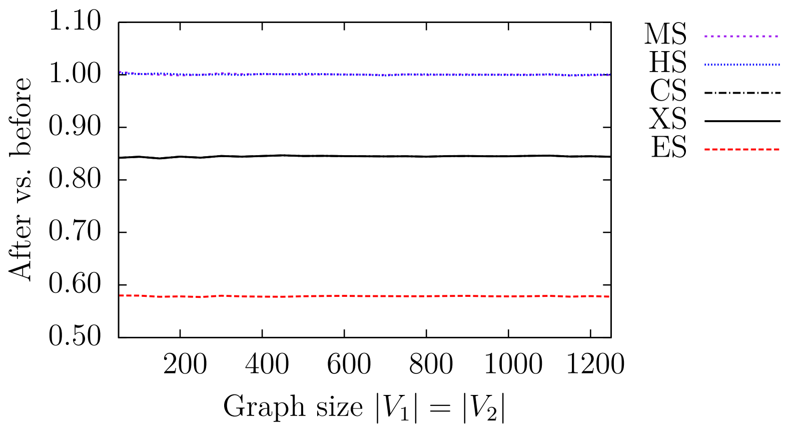


Fig. A.8. Benchmark: numbers of crossings between inter-level edges.

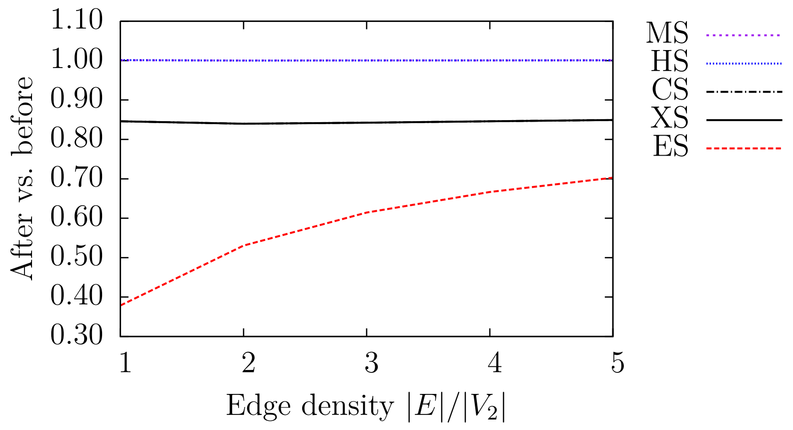


Fig. A.9. Benchmark: numbers of crossings between inter-level edges.

A.2 Concentric Level Lines

Figures A.10 to A.18 provide benchmark results comparing the heuristics to minimize crossings in radial drawings: inter-level sifting (ES) as in Sect. 4.1, intra-level sifting (HS) as in Sect. 4.2, mixed sifting (MS) as in Sect. 4.3.1, and extended sifting (XS) as in Sect. 4.3.2.

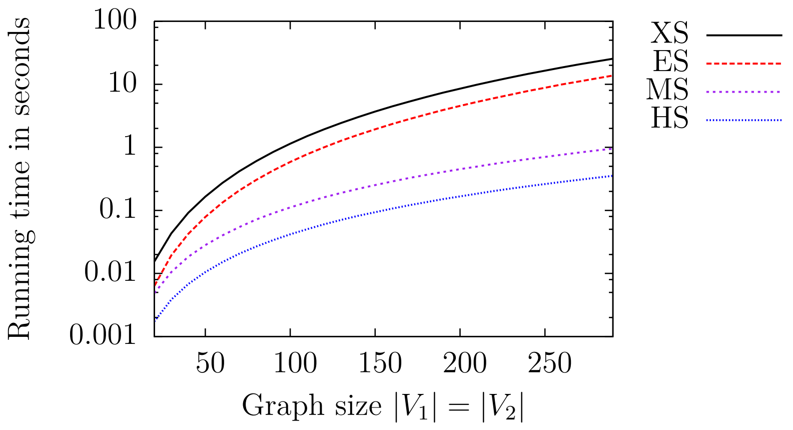


Fig. A.10. Benchmark: running times.

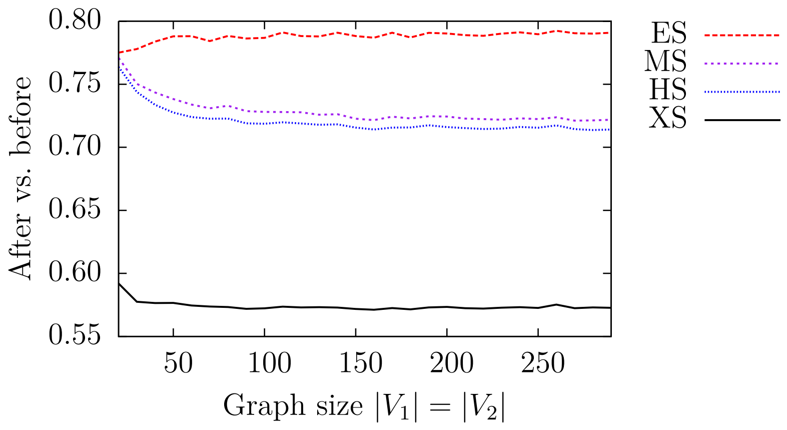


Fig. A.11. Benchmark: total crossing numbers.

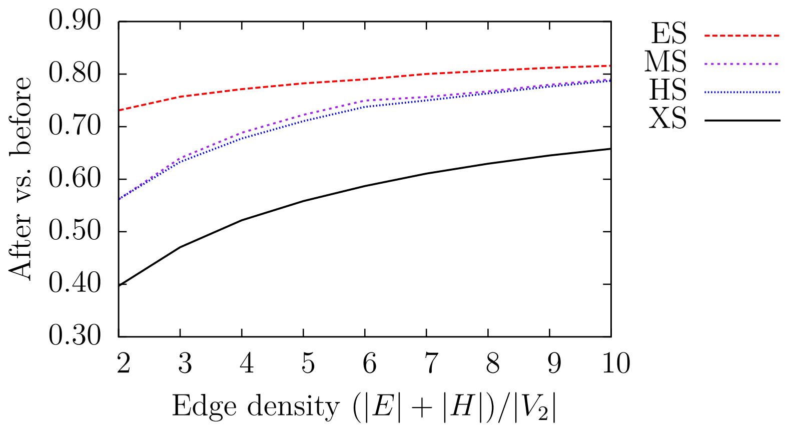


Fig. A.12. Benchmark: total crossing numbers.

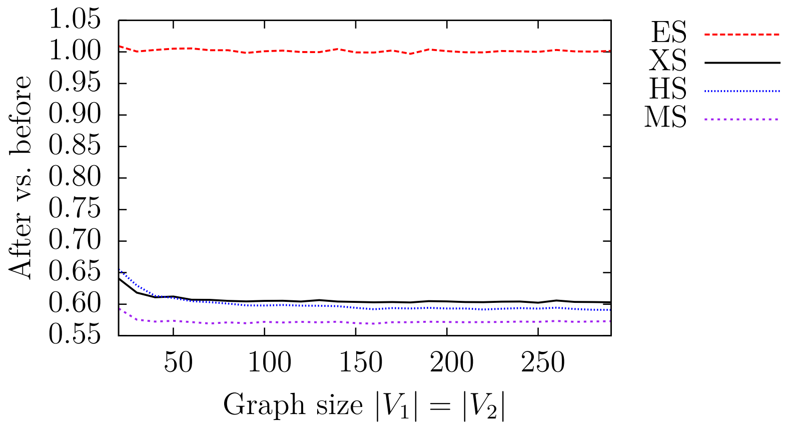


Fig. A.13. Benchmark: numbers of crossings between intra-level and inter-level edges.

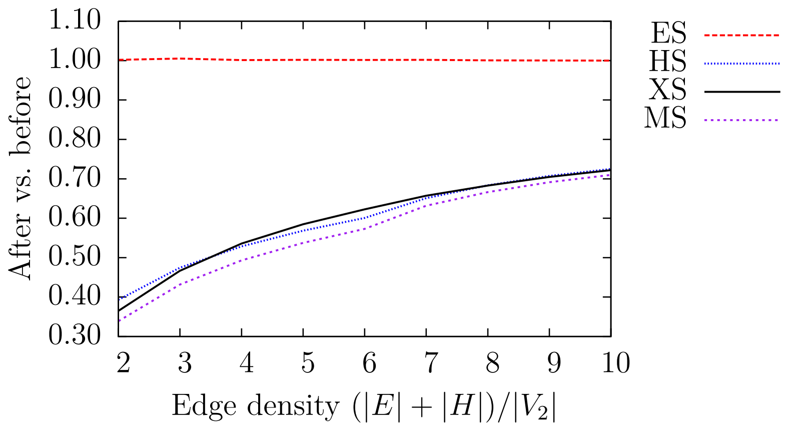


Fig. A.14. Benchmark: numbers of crossings between intra-level and inter-level edges.

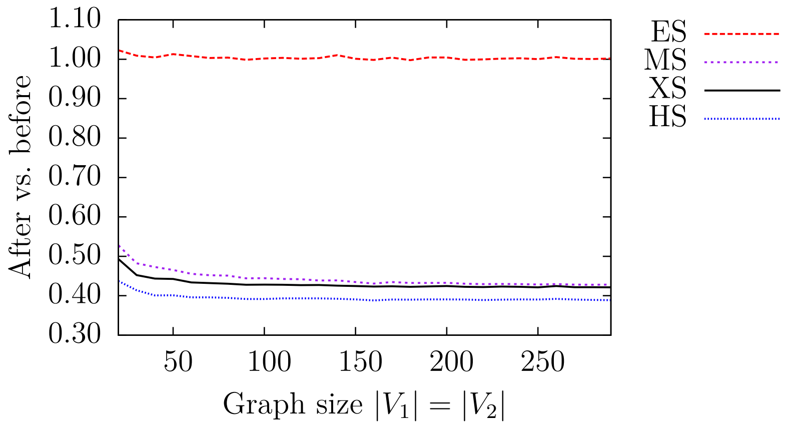


Fig. A.15. Benchmark: numbers of crossings between intra-level edges.

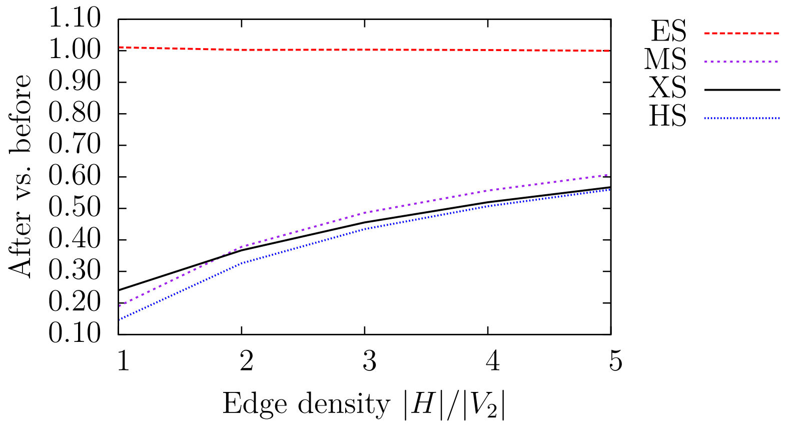


Fig. A.16. Benchmark: numbers of crossings between intra-level edges.

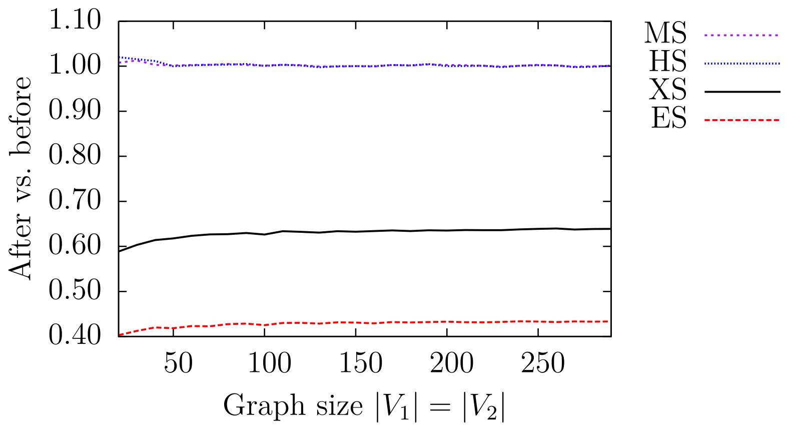


Fig. A.17. Benchmark: numbers of crossings between inter-level edges.

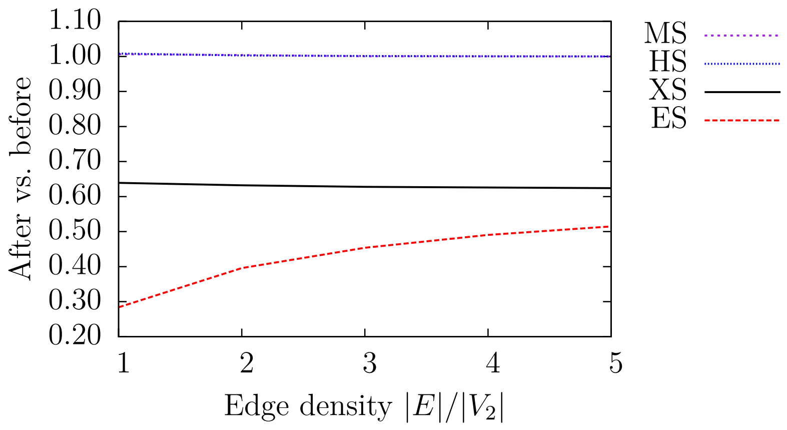
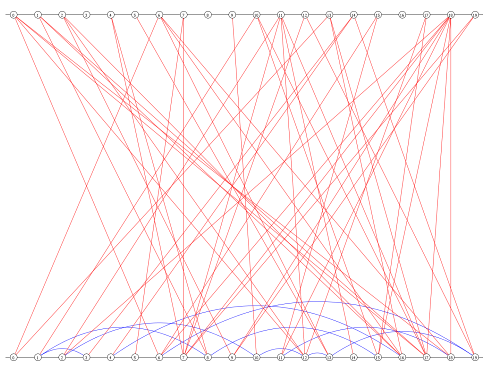


Fig. A.18. Benchmark: numbers of crossings between inter-level edges.

B Examples

Figures B.1 and B.2 show example outputs of both extended sifting algorithms. Remember, these are not results of a complete Sugiyama framework, as the fourth phase for the coordinate assignment is missing. Here, the vertices are uniformly distributed on their levels to show the orderings.


(a) Input with $\chi(\pi) = 768$.

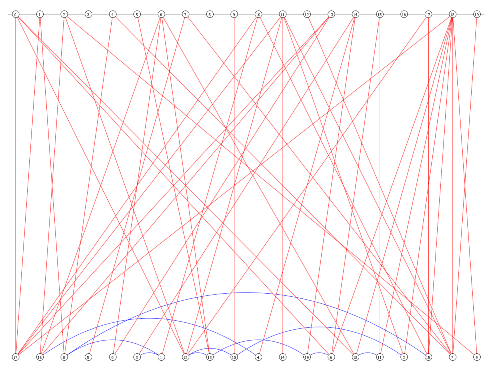
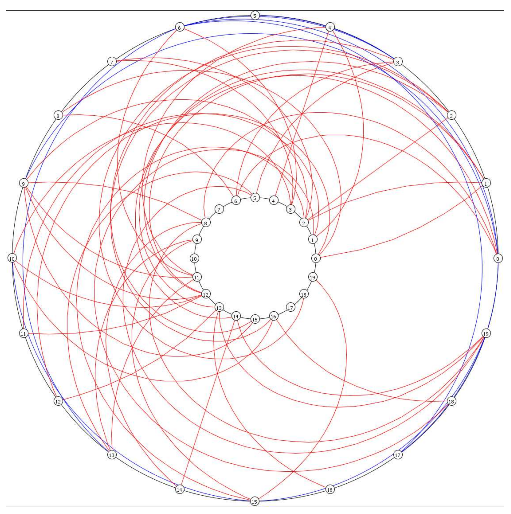

(b) Output after one sifting round with $\chi(\pi') = 441$.

Fig. B.1. Horizontal example with $|V_1| = |V_2| = 20$, $|E| = 50$, and $H = 10$.


(a) Input with $\chi(\pi) = 2405$.

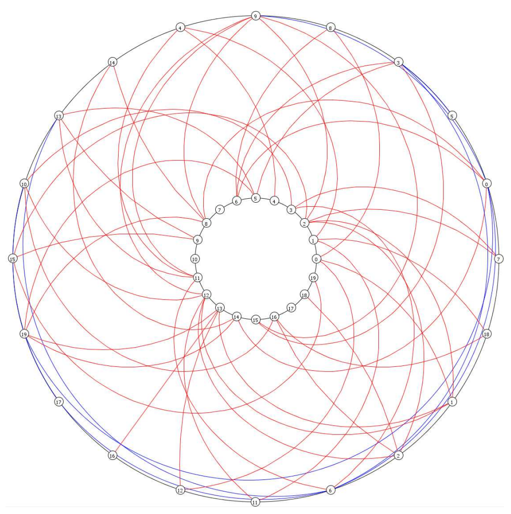

(b) Output one sifting round with $\chi(\pi') = 1397$.

Fig. B.2. Radial example with $|V_1| = |V_2| = 20$, $|E| = 50$, and $H = 10$.

A Global k-Level Crossing Reduction Algorithm[⋆]

Christian Bachmaier, Franz J. Brandenburg,
Wolfgang Brunner, and Ferdinand Hübner

University of Passau, Germany,
{bachmaier|brandenb|brunner|huebnerf}@fim.uni-passau.de

Abstract. Directed graphs are commonly drawn by the Sugiyama algorithm, where crossing reduction is a crucial phase. It is done by repeated one-sided 2-level crossing minimizations, which are still $\mathcal{NP}$-hard.
We introduce a global crossing reduction, which at any particular time captures all crossings, especially for long edges. Our approach is based on the sifting technique and improves the level-by-level heuristics in the hierarchic framework by a further reduction of the number of crossings by $5-10\%$. In addition it avoids type 2 conflicts which help to straighten the edges, and has a running time which is quadratic in the size of the input graph independently of dummy vertices. Finally, the approach can directly be extended to cyclic, radial, and clustered level graphs where it achieves similar improvements over the previous algorithms.

1 Introduction

The Sugiyama framework [12] is the standard drawing algorithm for directed graphs. It displays them in a hierarchical manner and operates in four phases: cycle removal (reverse appropriate edges to eliminate cycles), leveling (assign vertices to levels which define the y-coordinates and introduce dummy vertices on long edges), crossing reduction (permute the vertices on the levels), and coordinate assignment (assign x-coordinates to the vertices under some aesthetic criteria). Typical applications are schedules, UML diagrams, and flow charts.

In this paper we focus on the crossing reduction phase, where the vertices on each level are permuted to minimize the total number of crossings. The common solution for *k-level crossing minimization* is a reduction to the *one-sided 2-level crossing minimization problem*, which is solved repeatedly in some up and down sweeps [9, 12]. In the down sweep, the vertices V_{i-1} on the upper level are fixed and the vertices V_i of the lower level are reordered reducing the local number of edge crossings. In the up sweep the roles are switched. Even the one-sided 2-level crossing minimization problem is $\mathcal{NP}$-hard [6]. There are many heuristics for this problem [9]. Bastert and Matuszewski claim [9] that the results of this level-by-level sweep are far from optimum. “One can expect better results by considering all levels simultaneously, but k-level crossing minimization is a very hard problem” [9, page 102]. Our approach addresses this gap. Note that existing approximation ratios of 2-level algorithms do not translate to k-level graphs.

[⋆] Supported by the Deutsche Forschungsgemeinschaft (DFG), grant BR835/15-1.

An important feature of such algorithms is the guarantee of no *type 2 conflicts* which are crossings of two edges between dummy vertices. Among others, the standard fourth phase algorithm [4] by Brandes and Köpf assumes the absence of type 2 conflicts. Then it aligns long edges vertically and so achieves a crucial aesthetic criterion [9] for pleasing hierarchical drawings.

Common 2-level crossing reductions are the *barycenter* and *median heuristics* [9]. They place each vertex $v \in V_i$ in the barycenter or median position of its predecessors in V_{i-1}. After that V_i is sorted by these values. The idea is that on these positions the edges get short and, thus, generate few crossings. These techniques are simple, fast, and avoid type 2 conflicts, but leave many crossings.

Although such 2-level algorithms reduce the crossings between V_{i-1} and V_i, the number of crossings between V_i and V_{i+1} (and thus even the overall number of crossings) can increase while permuting V_i. These heuristics push the crossings downwards or upwards until they are resolved at level k or 1, respectively. An extension is *centered 3-level crossing reduction*, i. e., treating three consecutive levels V_{i-1}, V_i, V_{i+1} and permuting V_i while the orders of V_{i-1} and V_{i+1} are fixed s. t. the crossings between the three levels are reduced. However, this generates type 2 conflicts. For reaching a global optimum, all these algorithms are restricted to a local view. Thus, they may tend to get stuck in local optima.

Sifting was first used for vertex minimization in ordered binary decision diagrams [11] and later adapted to the one-sided 2-level crossing reduction [10]. The idea is to keep track of the number of crossings while in a *sifting step* a vertex $v \in V_i$ is moved along a fixed ordering of the vertices in V_i. Finally v is placed at its locally optimal position. The method is an extension of the greedy-switch heuristic [5], where v is swapped iteratively with its current successor. We call a single swap a *sifting swap* and the execution of a sifting step for every vertex in V_i a *sifting round*. Sifting leaves fewer crossings than the simple heuristics in general at the expense of a higher running time and potential type 2 conflicts [9].

Matuszewski et al. [10] have extended sifting towards a global view, which we call *ordered k-level sifting*. There the vertices are sorted by their degree and are sifted first in increasing order and then in decreasing order. All neighbors of the vertices to swap, i. e., on both neighboring levels, are considered. The heuristic does not sweep level-by-level but is still limited to a local view as long edges are not treated as a whole. Our *centered 3-level sifting* does the same level-by-level instead of ordered by degree. Both algorithms produce similar results. Jünger et al. [8] presented an exact ILP approach for the $\mathcal{NP}$-hard k-level crossing minimization, which can be used in practice for small graphs. Moreover, metaheuristics have been proposed in the literature, such as genetic algorithms, tabu search, or windows optimization.

In this paper we propose a new and global crossing reduction technique. The algorithm yields better results than traditional heuristics. It is easily extendable to more general crossing reduction problems, avoids type 2 conflicts, and runs in quadratic time in the size of the graph. Most 2-level approaches extensively use dummy vertices, whose number is up to $\mathcal{O}(k \cdot |E|) \subseteq \mathcal{O}(|V|^3)$ and do not make use of the edge bundling techniques of [7], which cannot be used for sifting.

2 Preliminaries

We suppose that a directed graph without self-loops has passed through the cycle removal and leveling phases. The outcome is a *k-level graph* $G = (V, E, \phi)$, where $\phi\colon V \to \{1, 2, \ldots, k\}$ is a surjective level assignment of the vertices with $\phi(u) < \phi(v)$ for each edge $(u, v) \in E$. For an edge $e = (u, v) \in E$ we define $\operatorname{span}(e) := \phi(v) - \phi(u)$. An edge e is *short* if $\operatorname{span}(e) = 1$ and *long* otherwise. A graph is *proper* if all edges are short. Each level graph can be made proper by adding $\operatorname{span}(e) - 1$ dummy vertices for each edge e which split e in $\operatorname{span}(e)$ many short edges. Let $G' = (V', E', \phi')$ denote the proper version of G. As in [4], short edges are called *segments* of e. The first and the last segments are the *outer* and the others the *inner segments*. Inner segments connect two dummy vertices.

For a vertex v we denote the set of neighbors from incoming and outgoing segments by $N^-(v) := \{\, u \in V' \mid (u, v) \in E' \,\}$ and $N^+(v) := \{\, w \in V' \mid (v, w) \in E' \,\}$, respectively. In an *ordered* proper level graph the vertices on each level as well as the sets $N^-(\cdot)$ and $N^+(\cdot)$ are ordered from left to right. Each proper level graph can be made ordered by choosing an arbitrary ordering for each level and sorting the sets $N^-(\cdot)$ and $N^+(\cdot)$ accordingly. In an ordered level graph there are two *conflicting* segments if they cross or share a vertex. Conflicts are of *type 0, 1* or *2*, if they are induced by 0, 1, or 2 inner segments, respectively.

Next we define blocks, which prevent dealing with dummy vertices and so keep the running time independent of them. A *block* is a single vertex of V or a maximum connected subgraph of dummy vertices, i. e., the inner segments of a long edge. The blocks represent the vertices of a graph related to G', where the edges are the outer segments. For a block A define $x = \operatorname{upper}(A)$ ($y = \operatorname{lower}(A)$) to be the unique vertex x in A (y in A) with no incoming (outgoing) segments in A. x and y always exist but may coincide. We define $N^-(A) := N^-(\operatorname{upper}(A))$, $N^+(A) := N^+(\operatorname{lower}(A))$, $\deg(A) := |N^-(A)| + |N^+(A)|$, and the set of all level numbers on which A has (dummy) vertices as $\operatorname{levels}(A)$. With $\operatorname{block}(v)$ we denote the block of the vertex $v \in V'$. Let $\mathcal{B}$ be any ordered list of all blocks and let $\pi\colon \mathcal{B} \to \{0, \ldots, |\mathcal{B}| - 1\}$ assign each block its current *position* in this ordering.

3 Global Sifting

A major drawback of the established crossing reduction algorithms is their local view. We present a new approach using ideas from [4] and [7][1] and avoiding type 2 conflicts. We treat all dummy vertices of an edge (and each original vertex) as one block and try to find the best position for the entire block in one step. This eliminates the problems of classic 2-level approaches which lack this global view on crossings of long edges. As an initialization the list of blocks $\mathcal{B}$ is sorted arbitrarily and each block A gets $\pi(A)$ as its position in $\mathcal{B}$ (line 1 in Algorithm 1). At any time during the execution of the algorithm interpreting $\pi(A)$ for each

[1] The authors of [7] use a data structure similar to our blocks and avoid type 2 conflicts. However, for crossing reduction they proceed level-by-level in the traditional fashion. Thus, only the running time but not the quality of the result is improved.

Algorithm 1: GLOBAL-SIFTING

```
Input: Proper $k$-level graph $G' = (V', E', \phi')$, number $\rho$ of sifting rounds
Output: Graph $G'$ with vertices ordered by values $\pi(v)$ for each $v \in V'$
1 create list $\mathcal{B}$ of all blocks in $G'$
2 for $1 \leq i \leq \rho$ do
3     foreach $A \in \mathcal{B}$ do
4         $\mathcal{B} \leftarrow$ SIFTING-STEP($G'$, $\mathcal{B}$, $A$)
5 foreach $v \in V'$ do $\pi(v) \leftarrow \pi(\text{block}(v))$
6 return $G'$
```

block A as an x-coordinate for each vertex v in A and $\phi(v)$ as its y-coordinate results in a drawing respecting the current ordering of $\mathcal{B}$. All vertices of a block get the same x-coordinate and, thus, the ordering is type 2 conflict free. These are important invariants of Algorithm 1.

The main part of the algorithm is the sifting step (line 4). There all positions for a block A are tested and A is moved to that position where it has the fewest crossings. This is done for each block $A \in \mathcal{B}$ (line 3) and repeated a certain number of times ρ (line 2). In practice, ten rounds suffice. Finally, each vertex is set to the position of its block (line 5) and the graph is returned (line 6).

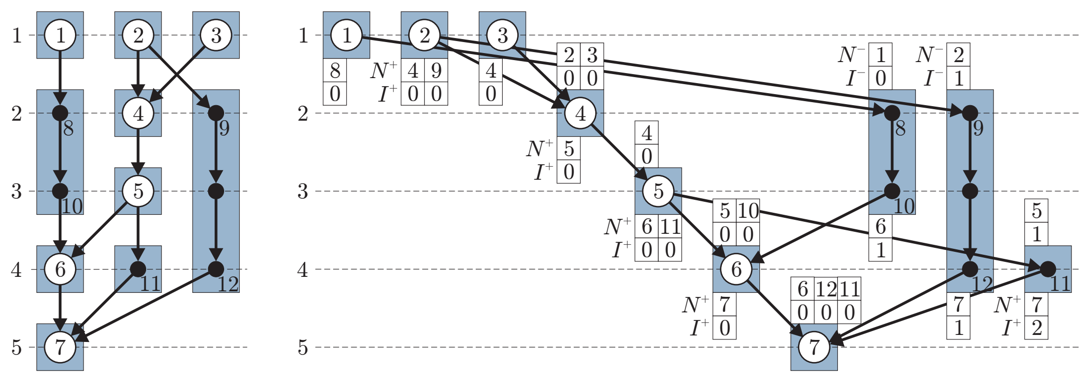
3.1 Building the Block List

The graph is partitioned into blocks. Each block A gets an arbitrary but unique position $\pi(A)$ in the block list $\mathcal{B}$. As an example consider Fig. 1(a). The input graph with 7 vertices gets 6 dummy vertices drawn as black circles. The dummy vertices are combined into 3 blocks and each original vertex forms its own block. The 10 resulting blocks are shown in Fig. 1(b) with an arbitrary ordering π.

If a given ordering should only be improved in a postprocessing step, a straightforward initialization strategy is to topologically sort the blocks according to the orderings on the levels from left to right in $\mathcal{O}(|E'|)$. Our experiments showed, that a good initial ordering of the blocks leads to better results. However, these can also be achieved by one or two additional sifting rounds.

3.2 Initialization of a Sifting Step

To improve the performance of one sifting step [3] it is necessary to keep the adjacency lists $N^-(A)$ and $N^+(A)$ of each block $A \in \mathcal{B}$ sorted according to ascending positions of the neighboring blocks in $\mathcal{B}$. We store them as arrays for random access. Additionally, we store two index arrays $I^-(A) = I^-(\text{upper}(A))$ and $I^+(A) = I^+(\text{lower}(A))$ of lengths $|I^-(A)| := |N^-(A)|$ and $|I^+(A)| := |N^+(A)|$, respectively. $I^-(A)$ stores the indices where upper(A) is stored in each adjacent block B's adjacency $N^+(B)$. More precisely, let $b = N^-(A)[i]$ be a neighbor of upper(A) with $B = \text{block}(b)$. Then $I^-(A)[i]$ holds the index at which upper(A)



(a) A level graph with ten blocks

(b) Separate π-positions, ordered adjacency lists N^- and N^+, and index arrays I^- and I^+ for each block

Fig. 1. Blocks as sifting objects

is stored in $N^+(B) = N^+(b)$. Symmetrically, $I^+(A)$ stores the indices at which lower(A) is stored in the adjacency $N^-(B)$ of each adjacent block B. See Fig. 1(b) for an example. The creation of the four arrays for each block (line 2 of Algorithm 3) can be done in $\mathcal{O}(|E|)$ time as Algorithm 2 shows: Traverse the blocks A in the current order of $\mathcal{B}$ and add upper(A) (lower(A)) to the next free position j of the cleared adjacency array $N^+(\text{lower}(B))$ ($N^-(\text{upper}(B))$) of each incoming (outgoing) neighbor B. Both values for $I^+(B)$ and $I^-(A)$ ($I^-(B)$ and $I^+(A)$) and their positions are only known after the second traversal of a segment s. Thus, we cache the first array position j as an attribute p of s. Benchmarks have shown that there is a considerable speed-up if only those adjacencies are updated that are no longer sorted after a sifting step. The theoretical running time is unaffected by this improvement.

Algorithm 2: SORT-ADJACENCIES

Input: Proper k-level graph $G' = (V', E', \phi')$, ordered list $\mathcal{B}$ of blocks in G'
Output: Ordered sets $N^{\cdot}(A)$ and $I^{\cdot}(A)$ for each block $A \in \mathcal{B}$

```
for $i \leftarrow 0$ to $|\mathcal{B}| - 1$ do $\pi(\mathcal{B}[i]) \leftarrow i$; clear arrays $N^{\cdot}(\mathcal{B}[i])$ and $I^{\cdot}(\mathcal{B}[i])$
foreach $A \in \mathcal{B}$ do
    foreach $s \in \{(u,v) \in E' \mid v = \text{upper}(A)\}$ do
        add $v$ to the next free position $j$ of $N^+(u)$
        if $\pi(A) < \pi(\text{block}(u))$ then $p[s] \leftarrow j$        // first traversal of s
        else $I^+(u)[j] \leftarrow p[s]$; $I^-(v)[p[s]] \leftarrow j$        // second traversal of s
    foreach $s \in \{(w,x) \in E' \mid w = \text{lower}(A)\}$ do
        add $w$ to the next free position $j$ of $N^-(x)$
        if $\pi(A) < \pi(\text{block}(x))$ then $p[s] \leftarrow j$        // first traversal of s
        else $I^-(x)[j] \leftarrow p[s]$; $I^+(w)[p[s]] \leftarrow j$        // second traversal of s
```

3.3 Sifting Step

In a sifting step (Algorithm 3) all positions p in $\mathcal{B}$ are tested for a block $A \in \mathcal{B}$ (lines 5–8) and then A is moved to the position p^* which has caused the least number of crossings. Note that it is not necessary to count the crossings for each position of A. As in [3] and contrary to classic sifting which always maintains the absolute number of crossings, we treat the number of crossings of A when put to the first position as $\chi = 0$. Then, we only compute the change in the number of crossings when iteratively swapping A with its right neighbor (line 6).

Algorithm 3: SIFTING-STEP

```
Input: Proper $k$-level graph $G' = (V', E', \phi')$, ordered list $\mathcal{B}$ of blocks in $G'$,
       block $A \in \mathcal{B}$ to sift
Output: Updated ordering of $\mathcal{B}$
1 $\mathcal{B}' \leftarrow A \prec \mathcal{B}[0] \prec \cdots \prec \mathcal{B}[|\mathcal{B}| - 1]$      // new ordering $\mathcal{B}'$ with $A$ put to front
2 SORT-ADJACENCIES($G'$, $\mathcal{B}'$)
3 $\chi \leftarrow 0$; $\chi^* \leftarrow 0$      // current and best number of crossings
4 $p^* \leftarrow 0$      // best block position
5 for $p \leftarrow 1$ to $|\mathcal{B}'| - 1$ do
6     $\chi \leftarrow \chi +$ SIFTING-SWAP($A, \mathcal{B}'[p]$)
7     if $\chi < \chi^*$ then
8         $\chi^* \leftarrow \chi$; $p^* \leftarrow p$
9 return $\mathcal{B}'[1] \prec \cdots \prec \mathcal{B}'[p^*] \prec A \prec \mathcal{B}'[p^* + 1] \prec \cdots \prec \mathcal{B}'[|\mathcal{B}'| - 1]$
```

3.4 Sifting Swap

The sifting swap is the actual computation of the change in the number of crossings when a block A is swapped with its right neighbor B. In contrast to one-sided crossing reduction, our global approach takes the whole neighborhood of both blocks into account when the change in the number of crossings is computed. Lemma 1 states which segments are involved.

Lemma 1. *Let $\mathcal{B}$ be the block list in the current ordering. Let $B \in \mathcal{B}$ be the successor of $A \in \mathcal{B}$. If swapping A and B changes the crossings between any two segments, then one of them is an incident outer segment of A or B. The other segment is an incident outer segment of the same kind (incoming or outgoing) of the other block or an inner segment of the other block.*

Proof. Note that only segments between the same levels can cross. As no type 2 conflicts occur at least one of the segments of a crossing has to be an outer segment. Let (a, b) and (c, d) be two segments between the same levels with $a \neq c$ and $b \neq d$. If the two segments cross after swapping A and B but did not cross before (or vice versa) either a and c or b and d were swapped. Therefore, one of the segments is adjacent to A or is a part of A and the other is adjacent to B or is a part of B. If b and d were swapped and thus a and c were not, $\phi(b) = \phi(d)$ is the upper level of A or B and thus one of the crossing segments is an incoming outer segment of A or B. The other segment is either an incoming outer segment or an inner segment of the other block. Note that it cannot be an outgoing outer segment of this block because then neither a and c nor b and d would have been swapped. The other case of swapping a and c instead of b and d is symmetric. □

Proposition 1. *Let $\mathcal{B}$ be the block list in the current ordering. Let $B \in \mathcal{B}$ be the successor of $A \in \mathcal{B}$. Let i and j be the two levels framing the incoming outer segments of A, the other three cases are symmetric. If there is a segment (u, v) between i and j which is either an incoming outer segment of B or an inner segment of B, then the incoming segments of A starting at a block left of* block(u) *cross (u, v) after the swap of A and B only, the segments starting at* block(u) *never cross (u, v), and the segments starting right of* block(u) *cross (u, v) before the swap only. There are no other changes of crossings due to Lemma 1.*

Algorithm 4 shows the details of a sifting swap. First, the levels at which (significant) swaps occur and the direction of the segments changing their crossings are found (lines 2–6). For each entry (l, d) of the set $\mathcal{L}$ the two vertices a and b of A and B on level l are retrieved. Note that when swapping A and B only a and b are swapped on their level and that in the level of their neighbors $N^d(a)$ and $N^d(b)$ no order changes. Thus, the computation of the change in the number of crossings can be done as in [3] (lines 14–24): The neighbors are traversed from left to right. If a neighbor of a is found (lines 19, 20) its segment will cross all remaining $s - j$ incident segments of b after the swap. If a neighbor of b is found (lines 21, 22) its segment has crossed all remaining $r - i$ incident segments of a before the swap. Common neighbors present both cases at the same time (line 23). An update of the adjacency after a swap (line 10) is only necessary if a and b have common neighbors. Algorithm 5 shows how this can be done in overall $\mathcal{O}(\deg(A) + \deg(B))$ time similarly to the crossing counting function `uswap`.

3.5 Time Complexity

Lemma 2. *Let $G = (V, E, \phi)$ be a level graph. Then $\sum_{B \in \mathcal{B}} \deg(B) \leq 4 \cdot |E|$.*

Proof. Each edge $e \in E$ contains at most two outer segments. Each outer segment increases the degree of its two incident blocks by one each. □

Theorem 1. *One round of global sifting (Algorithm 1) has a time complexity of $\mathcal{O}(|E|^2)$ for a non-necessarily proper level graph $G = (V, E, \phi)$.*

Proof. Let $\mathcal{B}$ be the blocks of G. Swapping two blocks $A, B \in \mathcal{B}$ needs $\mathcal{O}(\deg(A) + \deg(B))$ time. Initializing a sifting step takes $\mathcal{O}(\sum_{B \in \mathcal{B}} \deg(B)) = \mathcal{O}(|E|)$ time. A sifting step of a block A needs $\mathcal{O}(\sum_{B \in \mathcal{B} \setminus \{A\}} (\deg(A) + \deg(B))) = \mathcal{O}(|E| \cdot \deg(A))$ time. Thus, a sifting round positioning each block $A \in \mathcal{B}$ has time complexity $\mathcal{O}(\sum_{A \in \mathcal{B}} (|E| \cdot \deg(A)) = \mathcal{O}(|E|^2)$. Since $|V'| \leq k \cdot |E| \in \mathcal{O}(|E|^2)$ (no empty levels), traversing all (dummy) vertices in pre- and postprocessing has no effect on the worst case time complexity. □

Algorithm 4: SIFTING-SWAP

```
Input: Consecutive blocks $A, B$
Output: Change in crossing count
begin
    $\mathcal{L} \leftarrow \emptyset; \varDelta \leftarrow 0$
    if $\phi(\text{upper}(A)) \in \text{levels}(B)$ then $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\phi(\text{upper}(A), -)\}$
    if $\phi(\text{lower}(A)) \in \text{levels}(B)$ then $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\phi(\text{lower}(A), +)\}$
    if $\phi(\text{upper}(B)) \in \text{levels}(A)$ then $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\phi(\text{upper}(B), -)\}$
    if $\phi(\text{lower}(B)) \in \text{levels}(A)$ then $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\phi(\text{lower}(B), +)\}$
    foreach $(l, d) \in \mathcal{L}$ do
        let $a$ in $A$ and $b$ in $B$ be the vertices with $\phi(a) = \phi(b) = l$
        $\varDelta \leftarrow \varDelta + \mathsf{uswap}(a, b, N^d(a), N^d(b))$
        UPDATE-ADJACENCY$(a, b, N^d(a), I^d(a), N^d(b), I^d(b))$
    swap positions of $A$ and $B$ in $\mathcal{B}$; $\pi(A) \leftarrow \pi(A) + 1$; $\pi(B) \leftarrow \pi(B) - 1$
    return $\varDelta$
end

function uswap$(a, b, N^d(a), N^d(b))$: integer
    let $x_0 \prec \cdots \prec x_{r-1} \in N^d(a)$ be the neighbors of $a$ in direction $d$
    let $y_0 \prec \cdots \prec y_{s-1} \in N^d(b)$ be the neighbors of $b$ in direction $d$
    $c \leftarrow 0; i \leftarrow 0; j \leftarrow 0$
    while $i < r$ and $j < s$ do
        if $\pi(\text{block}(x_i)) < \pi(\text{block}(y_j))$ then
            $c \leftarrow c + (s - j)$; $i \leftarrow i + 1$
        else if $\pi(\text{block}(x_i)) > \pi(\text{block}(y_j))$ then
            $c \leftarrow c - (r - i)$; $j \leftarrow j + 1$
        else $c \leftarrow c + (s - j) - (r - i)$; $i \leftarrow i + 1; j \leftarrow j + 1$
    return $c$
```

Algorithm 5: UPDATE-ADJACENCIES

```
Input: Vertices $a, b \in V'$, $N^d(a), I^d(a), N^d(b), I^d(b)$
Output: Updated adjacencies of $a$ and $b$ and all common neighbors
let $x_0 \prec \cdots \prec x_{r-1} \in N^d(a)$ be the neighbors of $a$ in direction $d$
let $y_0 \prec \cdots \prec y_{s-1} \in N^d(b)$ be the neighbors of $b$ in direction $d$
$i \leftarrow 0; j \leftarrow 0$
while $i < r$ and $j < s$ do
    if $\pi(\text{block}(x_i)) < \pi(\text{block}(y_j))$ then $i \leftarrow i + 1$
    else if $\pi(\text{block}(x_i)) > \pi(\text{block}(y_j))$ then $j \leftarrow j + 1$
    else
        $z \leftarrow x_i$                                  // $= y_j$
        swap entries at positions $I^d(a)[i]$ and $I^d(b)[j]$ in $N^{-d}(z)$ and in $I^{-d}(z)$
        $I^d(a)[i] \leftarrow I^d(a)[i] + 1$; $I^d(b)[j] \leftarrow I^d(b)[j] - 1$
        $i \leftarrow i + 1; j \leftarrow j + 1$
```

4 Simple Global Crossing Reductions

We have extended the barycenter and median crossing reduction strategies towards blocks as well: We iteratively take the π-positions of the blocks in $\mathcal{B}$ and compute the barycenter or median for each block, respectively, and sort $\mathcal{B}$ according to these values. Our benchmarks show that both are very fast, however, are not competitive with global sifting in the number of crossings.

Theorem 2. *One round of global barycenter or global median has time complexity $\mathcal{O}(|E| \log |E|)$ or $\mathcal{O}(|E|)$, respectively.*

Proof. Computing the barycenters or medians for the $\mathcal{O}(|E|)$ blocks can be done in $\mathcal{O}(|E|)$ time due to Lemma 2. Sorting the barycenters takes $\mathcal{O}(|E| \log |E|)$ time. The medians can be sorted in $\mathcal{O}(|E|)$ time using bucket sort. □

5 Experimental Results

We have compared the iterative one-sided 2-level barycenter (B), median (M), and sifting (S), iterative centered 3-level sifting (3S), ordered k-level sifting (OS), and our new global barycenter (GB), global median (GM), and global sifting (GS) algorithms.

In a nutshell, classic sifting is fast, leaves few type 2 conflicts, but many crossings. Centered 3-level sifting is fast, leaves few crossings, but many type 2 conflicts. Global sifting leaves even less crossings (Fig. 2) without any type 2 conflicts within a still feasible running time in practice (Fig. 3). Further measurements reflect that the running time of global sifting is independent of the number of dummy vertices. This parallels the advanced algorithm in [7].

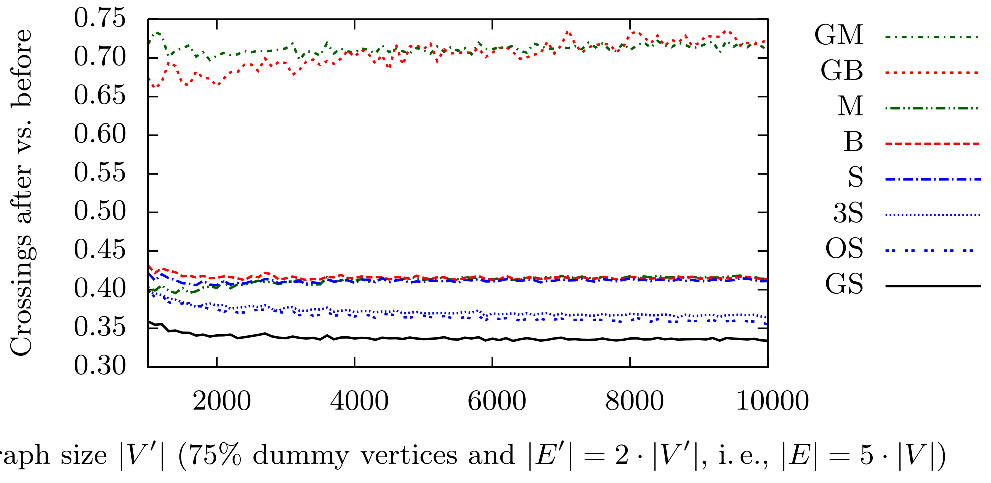


Fig. 2. Benchmark: number of crossings after vs. before applying the crossing reduction

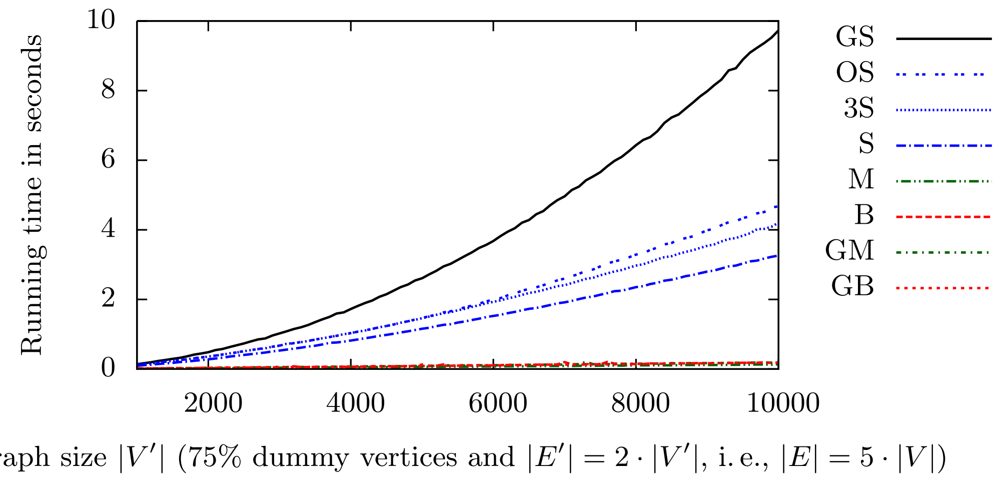


Fig. 3. Benchmark: running times

6 Applications of the Global Crossing Reduction

The idea of using blocks for long edges can be used in several other algorithms to improve their performance in a straightforward way. Further, this advances the drawability of their results as type 2 conflicts are avoided.

Optimal Crossing Reduction Using an ILP Jünger et al. [8] gave an ILP formulation for the exact crossing minimization of k-level graphs. Using pairs of overlapping blocks, i. e., on non-disjoint levels, as variables gives a direct formulation which naturally excludes type 2 conflicts and uses fewer variables.

Clustered Crossing Reduction In a clustered level graph vertices are combined to subgraphs in a hierarchical way. The crossing reduction has to ensure that all (dummy) vertices of a subgraph on the same level are consecutive and that all subgraphs spanning several levels have a matching ordering on each level to avoid crossings of subgraphs. This is rather complicated using a 2-level crossing reduction approach. Using global sifting this is quite simple: Instead of swapping a vertex with its right neighbor in a sifting swap we swap all blocks of a subgraph with its right neighbor (which itself is either a block or a subgraph) and determine the change in the number of crossings. The time complexity stays the same as in the normal global sifting. If the layout of the subgraphs themselves is not fixed, then global sifting can be applied to the subgraphs as well, e. g., performing a sifting round for each hierarchical layer.

Cyclic and Radial Level Graphs Level graphs can be extended to cyclic or radial level graphs. In cyclic level graphs the set of levels is ordered in a cyclic way, i. e., the first level follows the last one. In radial level graphs each level itself is ordered in a cyclic way, i. e., the first vertex on each level is the right neighbor of the last one. See Fig. 4 for clippings of drawings. For both, global sifting is the first crossing reduction to guarantee the needed absence of type 2 conflicts.

Cyclic levels are normally drawn forming a star in 2D (see Fig. 4(a)). These drawings explicitly visualize cycles in graphs [2], which is often required in bioinformatics. Our global sifting algorithm can be used for cyclic level graphs without any changes within the same time complexity. Note that one-sided 2-level algorithms cannot be applied here, since each of them pushes most of the crossings to the next level only. Even the absence of type 2 conflicts cannot be guaranteed then, because the sweep has to stop at some level.

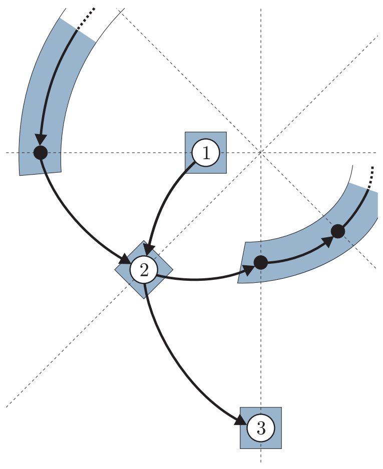
In a radial level graph the levels are concentric circles (see Fig. 4(b)). These drawings visualize distances or importance and are the traditional drawings of social networks. They map structural centrality of the graph to geometric centrality. Our global sifting approach guarantees radially aligned long edges and can be used with minor modifications: Each block of the block list $\mathcal{B}$ has its own angle. The ordering of $\mathcal{B}$ starts at an arbitrary block. Similar to [1], we define an *offset* $\psi : E \to \mathbb{Z}$ for each outer segment. The absolute value $|\psi(e)|$ counts the crossings of segment e with an imaginary *ray* splitting up the levels with a straight halfline from the concentric center to infinity. If $\psi(e) < 0$ ($\psi(e) > 0$), e has clockwise (counter-clockwise) direction read from source to target. When sifting a block $A \in \mathcal{B}$, we have to update the *partings*, which are the two borders between the counterclockwise and clockwise segments on the levels above and below A, see Fig. 4(b). Since we can do this independently of each other and add the results of the change in crossings to Δ in Algorithm 4, we use the same technique as in [1]. We sift a block from its current position in counterclockwise direction. Thus, for few crossings the partings have to follow in this direction on their levels. The test during the swap whether changing the orientations of some of the first of the (ordered) incident segments of A by incrementing their offsets, and thus putting them last, leads to less crossings and counting the difference raises the overall running time to $\mathcal{O}(|E|^3)$. The radial coordinate assignment phase in [1] relies on the obtained absence of type 2 conflicts.

7 Summary

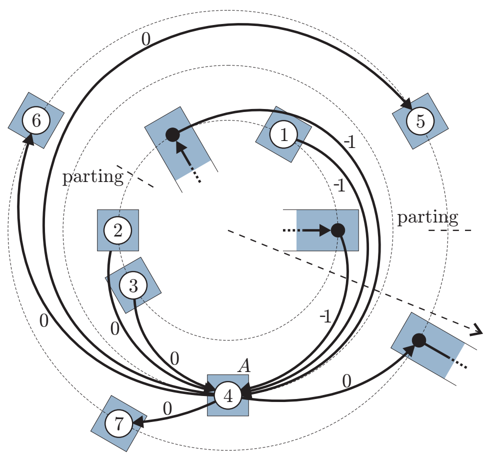
We have presented an algorithm for the global crossing reduction problem of k-level graphs. It produces high quality results with fewer crossings than common approaches at the expense of a quadratic running time, which is still feasible in practice. This was an open problem since the introduction of the hierarchical framework [12] in 1981. For cyclic and radial level crossing reduction we presented the first algorithms which guarantee the absence of type 2 conflicts. Our approach can easily be used to simplify and improve several other algorithms concerning level planarity or crossing reduction.

References

1. Bachmaier, C.: A radial adaption of the sugiyama framework for visualizing hierarchical information. IEEE Trans. Vis. Comput. Graphics 13(3), 583–594 (2007)



(a) Cyclic drawing



(b) Partings of the block A in a radial drawing

Fig. 4. Clippings of cyclic and radial drawings

2. Bachmaier, C., Brunner, W.: Linear time planarity testing and embedding of strongly connected cyclic level graphs. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 136–147. Springer (2008)
3. Baur, M., Brandes, U.: Crossing reduction in circular layout. In: Hromkovic, J., Nagl, M., Westfechtel, B. (eds.) Proc. Workshop on Graph-Theoretic Concepts in Computer Science, WG 2004. LNCS, vol. 3353, pp. 332–343. Springer (2004)
4. Brandes, U., Köpf, B.: Fast and simple horizontal coordinate assignment. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) GD 2001. LNCS, vol. 2265, pp. 31–44. Springer (2002)
5. Eades, P., Kelly, D.: Heuristics for reducing crossings in 2-layered networks. Ars Combinatorica 21(A), 89–98 (1986)
6. Eades, P., Wormald, N.C.: Edge crossings in drawings of bipartite graphs. Algorithmica 11(1), 379–403 (1994)
7. Eiglsperger, M., Siebenhaller, M., Kaufmann, M.: An efficient implementation of sugiyama's algorithm for layered graph drawing. J. Graph Alg. App. 9(3), 305–325 (2005)
8. Jünger, M., Lee, E.K., Mutzel, P., Odenthal, T.: A polyhedral approach to the multi-layer crossing minimization problem. In: Di Battista, G. (ed.) GD 1997. LNCS, vol. 1353, pp. 13–24. Springer (1997)
9. Kaufmann, M., Wagner, D.: Drawing Graphs, LNCS, vol. 2025. Springer (2001)
10. Matuszewski, C., Schönfeld, R., Molitor, P.: Using sifting for k-layer straightline crossing minimization. In: Kratochvíl, J. (ed.) GD 1999. LNCS, vol. 1731, pp. 217–224. Springer (1999)
11. Rudell, R.: Dynamic variable ordering for ordered binary decision diagrams. In: Proc. IEEE/ACM International Conference on Computer Aided Design, ICCAD 1993. pp. 42–47. IEEE Computer Society Press (1993)
12. Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Trans. Syst., Man, Cybern. 11(2), 109–125 (1981)

A.2 Articles for Sect. 3

Next we reprint [2].

IEEE TRANSCATIONS ON VISUALIZATION AND COMPUTER GRAPHICS, VOL. 13, NO. 3, MAI/JUNE 2007 1

A Radial Adaptation of the Sugiyama Framework for Visualizing Hierarchical Information

Christian Bachmaier

***Abstract*— In radial drawings of hierarchical graphs the vertices are placed on concentric circles rather than on horizontal lines and the edges are drawn as outwards monotone segments of spirals rather than straight lines as it is both done in the standard Sugiyama framework. This drawing style is well suited for the visualisation of centrality in social networks and similar concepts. Radial drawings also allow a more flexible edge routing than horizontal drawings, as edges can be routed around the center in two directions. In experimental results this reduces the number of crossings by approximately 30 percent on average. Few crossings are one of the major criteria for human readability.**

This paper is a detailed description of a complete framework for visualizing hierarchical information in a new radial fashion. Particularly, we briefly cover extensions of the level assignment step to benefit by the increasing perimeters of the circles, present three heuristics for crossing reduction in radial level drawings, and also show how to visualize the results.

***Index Terms*— graph drawing, radial, crossing reduction, Sugiyama framework, spiral segments**

I. Introduction

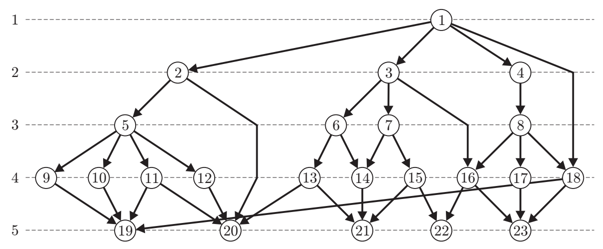
HIERARCHICAL graph layout is the method of choice for visualizing a general direction of flow, e. g., data or information, in relational data. Thereby vertices are usually drawn on parallel horizontal lines, and edges are drawn as y-monotone polylines that may bend when they intersect a level line. The standard drawing algorithm [1] for visualizing flow in vertical direction consists of four phases: *cycle removal* (reverses appropriate edges to eliminate cycles), *level assignment* (assigns vertices to levels and introduces dummy vertices to represent edge bends), *crossing reduction* (permutes vertices on the levels), and *coordinate assignment* (assigns x-coordinates to vertices, y-coordinates are implicit through the levels). See [2] for an extended overview.

The novelty of this paper is that we draw the level lines as concentric circles instead of as parallel horizontal lines and thus visualize flow from the center outwards. Further, in analogy to y-monotone straight line edges used in horizontal level drawings, we draw the edges as segments of spirals, unless they are radially aligned, in which case they are drawn as straight lines. This results in strictly monotone curves from inner to outer levels and ensures that the segments do not cross inner level lines or each other unnecessarily. The apparent advantage of radial level drawings is that level graphs can be drawn with fewer edge crossings. It is also more likely that a graph can be drawn without any crossings at all, since the set of level planar graphs is a proper subset of the set of radial level planar graphs [3]. Note that radial level drawings are different from circular drawings [4]–[6] where only one circle contains all vertices and do not comply with radial drawings [7] where edges are drawn straight line and level lines are not equidistant. Further, in contrast to both, here “inner level edges” with both end vertices on a common level are prohibited. See Fig. 1 for an example.

Radial drawings are not new, for example see the hierarchical graph drawings of [8], the ring diagrams of [9], or the radial tree drawings of [10], which are common for free trees. Radial level drawings are also common, e. g., in the study of social networks [11], [12]. There vertices model *actors* and edges represent *relations* between the actors. The importance (*centrality*) of a vertex is expressed by its distance (*closeness*) to the center, i. e., a position on a low level. Radial level drawings are also well suited for level graphs with an increasing number of vertices on higher levels. For example, in a graph that shows which Web pages are reachable from a given start page by following k hyperlinks, higher levels are likely to contain many vertices while there are only few vertices on the lower

Manuscript received 14 June 2006; revised 11 Oct. 2006; accepted 19 Oct. 2006; published online 2 Jan. 2007.

The author is with the Fakultät für Mathematik und Informatik, Universtät Passau, Innstr. 33, 94032 Passau, Germany. E-mail: bachmaier@fmi.uni-passau.de


(a) Horizontal

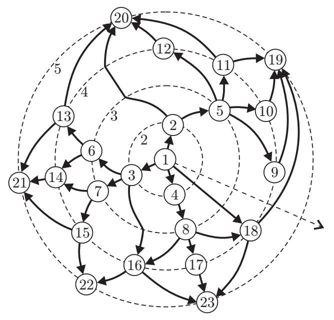

(b) Radial

Fig. 1. Drawings of a level graph

levels. Other potential applications may be found in [13], [14].

The paper is organized as follows: After some preliminary definitions in the next section, we present a complete framework in Sugiyama style to create radial level drawings of hierarchical graphs. This is done by introducing methods for radial level assignment in Sect. III, radial crossing reduction in Sect. IV, and radial coordinate assignment [15] in Sect. V. We omit the cycle removal step since it does not differ from the horizontal case and standard algorithms can be used. See [2] for an overview on that topic. We conclude in Sect. VI.

II. Preliminaries

A *k-level graph* $G = (V, E, \phi)$ is a directed acyclic graph (DAG) with a level assignment $\phi\colon V \to \{1, 2, \ldots, k\}$, which partitions the vertex set into $k \le |V|$ pairwise disjoint subsets $V = V_1 \dot\cup V_2 \dot\cup \cdots \dot\cup V_k$, $V_i = \phi^{-1}(i)$, $1 \le i \le k$, such that $\phi(u) < \phi(v)$ for each edge $(u, v) \in E$. Particularly, $k = 1$ implies that $E = \emptyset$. An edge (u, v) is *short* if $\phi(v) - \phi(u) = 1$; otherwise it is *long* and *spans* the levels $\phi(u)+1, \ldots, \phi(v)-1$. A level graph without long edges is *proper*. Any level graph can be made proper by subdividing each long edge $(u, v) \in E$ by the introduction of new *dummy vertices* $v_i \in B_i$, $i = \phi(u) + 1, \ldots, \phi(v) - 1$, $\phi(v_i) = i$. We draw dummy vertices as small black circles, Fig. 6(a), or edge bends, Fig. 1. The edges of a proper level graph are also called (edge) *segments*. If both end vertices of a segment are dummy vertices, it is called an *inner segment*. Let $N = |V \cup B| + |B| + |E|$ denote the size of the proper level graph $G = (V \cup B, E, \phi)$ where V contains the original vertices and $B = B_1 \dot\cup B_2 \dot\cup \ldots \dot\cup B_k$ contains the dummy vertices with $|B| \le k|E| \le |V||E|$.

An *ordering* of a proper level graph is a partial order $\prec$ of $V \cup B$ such that $u \prec v$ or $v \prec u$ iff $\phi(u) = \phi(v)$. This is equivalent to a definition of the vertex *positions* on level i as a bijective function $\pi_i\colon V_i \cup B_i \to \{0, \ldots, |V_i| + |B_i| - 1\}$ with $u \prec v \Leftrightarrow \pi_i(u) < \pi_i(v)$ for any two vertices $u, v \in V_i \cup B_i$. Let $\pi = (\pi_i)_{1 \le i \le k}$. We call an ordering of a level graph a *horizontal embedding*. In case that an ordering admits a drawing without edge crossings (except common end points), it is called a *level planar embedding*. Throughout the paper let $N^-(v) = \{\, u \mid (u, v) \in E \,\}$ denote the predecessors of $v \in V$. A vertex is called a *source* (*sink*), if there is no incoming (outgoing) incident edge. Let $\operatorname{sgn}\colon \mathbb{R} \to \{-1, 0, 1\}$ denote the signum function.

III. Radial Level Assignment

The basic problem is the same as in horizontal level drawings: In this phase a given DAG is to be transformed into a level graph by assigning the vertices to levels. Thus, any existing level assignment algorithm for horizontal level drawings can directly be used for radial level drawings. The optimization criteria, however, slightly change: Radial level drawings use k concentric circles to place the vertices of the k levels. Contrary to the constant line lengths in horizontal level drawings, the perimeters of the circles grow longer with ascending level numbers: On an outer circle, there is space for more vertices than on an inner circle. Thus, we investigate how level assignment methods can be extended to take advantage of this.

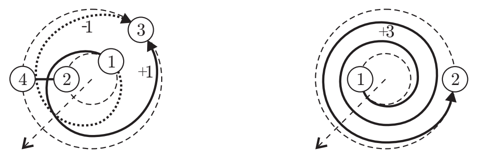
A straight-forward idea is to apply the longest path level assignment method from outer to inner levels: First, each sink of the graph is assigned to the highest level. For the remaining vertices the level is recursively defined by $\phi(v) = \min\{\, \phi(w) \mid (v, w) \in E \,\} - 1$. This puts each vertex on the outermost possible level while minimizing the number of levels k. There is no explicit balancing of level sizes, however.

For a better vertex distribution, an extension of the Coffman/Graham algorithm [16] can be used that explicitly takes into account the growing perimeter of the circles. Coffman and Graham compute a leveling where the number of vertices per level is bounded by a given constant W. We change this bound to be a function $w(i)$ which grows proportionally with the index i of a level: $w(i) = W \cdot i$. The first phase of the algorithm remains unchanged, but we apply it using the opposite edge direction: After removing transitive edges in linear time, an appropriate numbering $o\colon V \to \{0, \dots, |V| - 1\}$ of the vertices is computed: Initially, all vertices are unnumbered. We consecutively choose one vertex at a time and assign the next ascending number to it. The vertex is chosen so that it has no unnumbered successors and that the numbers of the successors are minimal regarding a specific ordering of integer sets: a set of vertex numbers is considered less than another one, if the maximum is less. If the maximum of both sets is equal, the next smaller value is compared, and so on. In the second phase the algorithm places one vertex at a time, starting with the vertex numbered with $|V|-1$ on level $i = 1$ and filling the levels from inner to outer circles. In one step it places the next unleveled vertex $v \in V$ with maximum $o(v)$ whose predecessors are already leveled. If level i is full, i. e., if i contains $w(i)$ vertices, or if v has a predecessor u with $\phi(u) = i$, then a new level is started, i. e., i is increased by 1. The level of v is then set to $\phi(v) = i$.

As the last step of the level assignment phase the level graph is made proper, because for drawing level graphs it is necessary to know where long edges should be routed, i. e., between which two vertices on a spanned level. Thus all long edges are subdivided in proper segments by new dummy vertices B in $\mathcal{O}(k|E|)$ running time. In the following, we will only consider proper level graphs.

IV. Radial Crossing Reduction

REGARDLESS of whether the leveling of a level graph is given by the application or if it has been computed by one of the algorithms in the previous section, the next step towards a hierarchical drawing is to compute an embedding. In horizontal hierarchical drawings the embedding is fully defined by the vertex ordering π. For radial embeddings it is also necessary to know where the (w. l. o. g. counter-clockwise) orderings start and end on each level. Therefore, we introduce a *ray* that tags this borderline between the vertices, cf. Fig. 1(b). The ray is a straight halfline from the center to infinity between the vertices on each level with extremal positions. Edges crossed by the ray are called *cut edges*. In horizontal drawings of level graphs a crossing between two edges only depends



(a) Edge $(1,3)$ drawn counter-clockwise and clockwise (dotted)

(b) $\psi\left((1,2)\right) = +3$

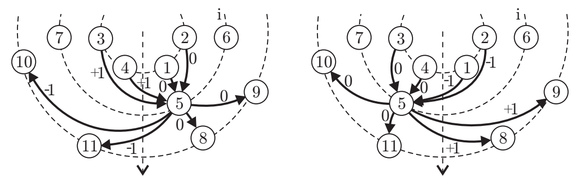
Fig. 2. Offsets of edges

on the orderings of the end vertices. In radial level drawings, however, it is also necessary to consider the direction in which the edges are wound around the center, clockwise or counter-clockwise. Furthermore, edges can also be wound around the center multiple times. We call this the offset $\psi\colon E \to \mathbb{Z}$ of an edge. Thereby, $|\psi(e)|$ counts the crossings of an edge $e \in E$ with the ray. If $\psi(e) < 0$ ($\psi(e) > 0$), e is a *clockwise* (*counter-clockwise*) cut edge, i. e., the sign of $\psi(e)$ reflects the mathematical direction of rotation, see Fig. 2. If $\psi(e) = 0$, then e is not a cut edge and thus needs no direction information. Observe that a cut edge cannot cross the ray clockwise and counter-clockwise simultaneously. We define a *radial embedding* $\mathcal{E}$ of a graph $G = (V, E, \phi)$[1] to consist of the vertex ordering π and of the edge offsets ψ, i. e., $\mathcal{E} = (\pi, \psi)$.

Compared to horizontal drawings there is an additional freedom in radial drawings without changing the crossing number: *rotation* of a level i. A clockwise rotation moves the vertex v with minimum position on the ordered level $\phi(v) = i$ over the ray by setting $\pi_i(v)$ to the maximum on i. The other values of π_i are updated accordingly. For an illustration see Fig. 3, where $v = 5$. A counter-clockwise rotation is defined symmetrically. Rotations do not modify the "cyclic order", i. e., the neighborhood of every vertex on its radial level line is preserved. However, the offsets of the edges incident to v must be updated. If rotating clockwise, the offsets of incoming edges of v are reduced by 1 and the offsets of outgoing edges are increased by 1. The offset updates for rotating counter-clockwise are symmetric. Depending on the implementation, rotation needs $\mathcal{O}(\deg(v))$ resp. $\mathcal{O}(|V| + \deg(v))$ running time.

The most common technique for crossing reduction in proper level graphs is to only consider two consecu-

[1]We need not to distinguish between original and dummy vertices in this phase. Thus here V denotes both of them for an easy notation.



(a) Radial embedding (b) Clockwise rotation of level i

Fig. 3. Rotation

tive levels at a time in multiple top-down and bottom-up passes. Starting with an arbitrary permutation of the first level, subsequently the ordering of one level is fixed, while the next one is reordered to minimize the number of crossings in-between. This *one-sided two-level crossing reduction problem* is $\mathcal{NP}$-hard [17] and well-studied. For radial level embeddings we follow the same strategy and consider the *radial one-sided two-level crossing reduction problem.* Given a 2-level graph $G = (V_1 \dot\cup V_2, E, \phi)$ and an ordering π_1 of the first level, our objective is to compute an ordering of the second level and offsets for the edges with few crossings.

A. Properties

Crossings between edges in radial embeddings depend on their offsets and on the order of the end vertices. There can be more than one crossing between two edges, if they have very different offsets. We denote the number of crossings between two edges $e_1, e_2 \in E$ in an embedding $\mathcal{E}$ by $\chi_\mathcal{E}(e_1, e_2)$. Intuitively, this number is approximately equal to the difference of the offsets $|\psi(e_2) - \psi(e_1)|$. The exact formula is slightly different, however, with a small shift depending on the vertex ordering, see Lemma 1.[2] The (radial) crossing numbers of a radial embedding $\mathcal{E}$ and of a level graph $G = (V, E, \phi)$ are then naturally defined as $\chi(\mathcal{E}) = \sum_{\{e_1,e_2\} \subseteq E, e_1 \neq e_2} \chi_\mathcal{E}(e_1, e_2)$ and $\chi(G) = \min\{\chi(\mathcal{E}) \mid \mathcal{E} \text{ is a radial embedding of } G\}$.

Lemma 1: Let $\mathcal{E} = (\pi, \psi)$ be a radial embedding of a 2-level graph $G = (V_1 \dot\cup V_2, E, \phi)$. Then the number of crossings between two edges $e_1 = (u_1, v_1) \in E$ and $e_2 = (u_2, v_2) \in E$ is

$$\chi_\mathcal{E}(e_1, e_2) = \\ \max\left\{0, \left|\psi(e_2) - \psi(e_1) + \tfrac{b-a}{2}\right| + \tfrac{|a|+|b|}{2} - 1\right\},$$

TABLE I

THE CROSSING NUMBER IN RELATION TO $\delta = \psi(e_2) - \psi(e_1)$

u_1 vs. u_2	v_1 vs. v_2	$\psi(e_2)$ vs. $\psi(e_1)$	$\chi_\mathcal{E}(e_1, e_2)$	u_1 vs. u_2	v_1 vs. v_2	$\psi(e_2)$ vs. $\psi(e_1)$	$\chi_\mathcal{E}(e_1, e_2)$
$\prec$	$\prec$	$<$	$\lvert\delta\rvert$	$\succ$	$\succ$	$>$	$\lvert\delta\rvert$
$\prec$	$\prec$	$=$	0	$\succ$	$\succ$	$=$	0
$\prec$	$\prec$	$>$	$\lvert\delta\rvert$	$\succ$	$\succ$	$<$	$\lvert\delta\rvert$
$\prec$	$=$	$<$	$\lvert\delta - \frac{1}{2}\rvert - \frac{1}{2}$	$\succ$	$=$	$>$	$\lvert\delta + \frac{1}{2}\rvert - \frac{1}{2}$
$\prec$	$=$	$=$	0	$\succ$	$=$	$=$	0
$\prec$	$=$	$>$	$\lvert\delta - \frac{1}{2}\rvert - \frac{1}{2}$	$\succ$	$=$	$<$	$\lvert\delta + \frac{1}{2}\rvert - \frac{1}{2}$
$\prec$	$\succ$	$<$	$\lvert\delta - 1\rvert$	$\succ$	$\prec$	$>$	$\lvert\delta\rvert + 1$
$\prec$	$\succ$	$=$	1	$\succ$	$\prec$	$=$	1
$\prec$	$\succ$	$>$	$\lvert\delta - 1\rvert$	$\succ$	$\prec$	$<$	$\lvert\delta\rvert + 1$
$=$	$\prec$	$<$	$\lvert\delta - \frac{1}{2}\rvert - \frac{1}{2}$	$=$	$\succ$	$>$	$\lvert\delta - \frac{1}{2}\rvert - \frac{1}{2}$
$=$	$\prec$	$=$	0	$=$	$\succ$	$=$	0
$=$	$\prec$	$>$	$\lvert\delta - \frac{1}{2}\rvert - \frac{1}{2}$	$=$	$\succ$	$<$	$\lvert\delta - \frac{1}{2}\rvert - \frac{1}{2}$
$=$	$=$	$<$	$\lvert\delta\rvert - 1$	$=$	$=$	$>$	$\lvert\delta\rvert - 1$
$=$	$=$	$=$	0				

where $a = \operatorname{sgn}(\pi_1(u_2) - \pi_1(u_1))$ and

$$b = \operatorname{sgn}(\pi_2(v_2) - \pi_2(v_1)) \ .$$

Proof: In analogy to horizontal embeddings, edge crossings do not depend on the exact position of the end vertices, but only on the relative ordering ($\prec$, $\succ$, or $=$) and on the edge offsets. We can assume w. l. o. g. that $\psi(e_1) = 0$, because in any embedding we can rotate the whole second level multiple times without changing π_2 or the offset difference $\delta = \psi(e_2) - \psi(e_1)$. This leads to $3 \cdot 3 \cdot 3 = 27$ cases, which are straightforward to prove, see Tab. IV-A. ■

Corollary 1: Let $\mathcal{E}$ be a radial embedding of a 2-level graph $G = (V_1 \dot\cup V_2, E, \phi)$. Swapping the position of two vertices $v, w \in V_2$ changes the number of crossing between two edges $(\cdot, v), (\cdot, w) \in E$ by at most 1.

Before we show our radial crossing reduction algorithms, we first discuss some more properties which follow from radial level lines.

Lemma 2: Let $G = (V_1 \dot\cup V_2, E, \phi)$ be a 2-level graph and let $e_1 = (u_1, v) \in E$ and $e_2 = (u_2, v) \in E$ be two edges with a common target vertex v. Then in any crossing minimal radial embedding $\mathcal{E} = (\pi, \psi)$ of G, $\pi_1(u_1) < \pi_1(u_2)$ implies $\psi(e_2) - \psi(e_1) \in \{0, 1\}$.

Proof: Assume that $\psi(e_2) - \psi(e_1) \notin \{0, 1\}$. Then Lemma 1 implies $\chi_\mathcal{E}(e_1, e_2) > 0$. We choose an arbitrary crossing between e_1 and e_2 and show how the embedding can be modified to reduce the number of crossings, see Fig. 4(a) for an illustration. We exchange the routing of e_1 and e_2 between v and the crossing: e_1 is routed along the old course of e_2 until it reaches the crossing. The routing from there to u_1 is not changed. We symmetrically do the same

[2]Although high offsets are never useful for a low number of crossings, we nevertheless provide the general result, not only to show that it also can be computed in constant time, but also since it is an interesting problem in itself.

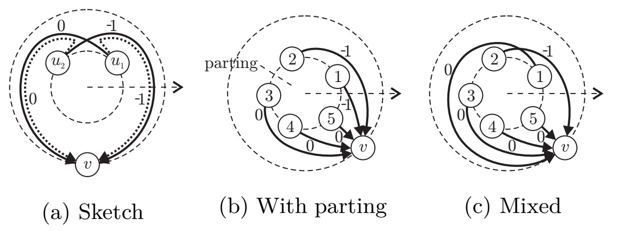


Fig. 4. Not all offset combinations for edges $(\cdot, v) \in E$ result in few crossings

with e_2. In the new embedding e_1 and e_2 have one crossing less and the number of crossings has not changed otherwise, contradicting the assumption and proving the lemma. ■

Because of this result, it is clear that only embeddings need to be considered, where there is a clear *parting* between all edges incident to the same vertex as in Fig. 4(b). The parting is that position of the edge list of v that separates the two subsequences with offsets ψ_0 resp. $\psi_0 + 1$. Otherwise unnecessary crossings are generated between the incident edges, see Fig. 4(c). We also only consider radial embeddings with small edge offsets, because large offsets correspond to very long edges which are difficult to follow. Obviously, winding edges more than once around the center can only increase its number of crossings. Thus, only the offsets -1, 0, and 1 are used in our algorithms.

Lemma 3: Radial one-sided two-level crossing minimization is $\mathcal{NP}$-hard.

Proof: We show the $\mathcal{NP}$-hardness by reduction from the horizontal one-sided two-level crossing minimization problem, which is known to be $\mathcal{NP}$-hard [17]. Given a 2-level graph $G = (V_1 \dot\cup V_2, E, \phi)$ with a fixed permutation π_1 of the first level, we construct a new 2-level graph $G' = (V_1' \dot\cup V_2', E', \phi')$ by adding a barrier as follows: G is extended by $|E|^2$ new vertices $x_0, \dots, x_{|E|^2-1}$ at the end of the first level $\pi_1'(x_i) = |V_1| + i$, $\phi'(x_i) = 1$ and a new vertex y on the second level $\phi'(y) = 2$ that is connected to them by new edges $e_0, \dots, e_{|E|^2-1}$, $e_i = (x_i, y)$.

Let $\mathcal{E}' = ((\pi_1', \pi_2'), \psi)$ be a radial embedding of G' that has a minimum number of crossings subject to π_1'. We can assume w. l. o. g. (because of rotation and Lemma 2) that $\pi_2'(y) = |V_2|$ and $\psi(e_i) = 0$ for all new edges. Then none of the new edges has a crossing with any of the original edges, because this would lead to $|E|^2$ crossings, contradicting the minimality of the embedding. Thus, there are no cut edges, and $\pi_2 = \pi_2'|_{V_2}$ is a solution of the original horizontal one-sided two-level crossing minimization problem. ■

As a consequence, we use heuristics for an efficient solution of the problem. In the following, we present three different approaches, extending some well-known [2] horizontal one-sided two-level crossing reduction methods, namely the barycenter, median, and sifting heuristics.

B. Cartesian Barycenter Heuristic

In the horizontal barycenter crossing reduction method for every vertex in V_2 the average value of the positions of its neighbors in V_1 is computed. Afterward V_2 is sorted according to this values following the rule of thumb "shorter edges have less crossings than longer edges". With some restrictions, this method can be directly used to compute a radial embedding: The horizontal vertex ordering defines a radial vertex ordering, and all edge offsets are set to 0. This neglects the additional freedom of radial edge routing, however, and therefore introduces more crossings than necessary. The result is especially bad for vertices whose neighbors on the first level are far apart. If, for an extreme example, a vertex is only adjacent to the first and last vertex of the first level, its best position is obviously near the ray, labeling one of the edges as a cut edge. But the horizontal algorithm cannot do that, and therefore produces an out-of-balance embedding. Even worse, the result depends on the current position of the ray.

One approach to improve that could be to rotate the first level before computing the barycenter values to an appropriate position, or maybe even use different rotations for different vertices. We propose a simpler, yet equally promising method. The basic idea stays the same: each vertex should be close to the average position of its neighbors. However, we use the terms "average" and "position" in a geometric sense. We assume the vertices of the first level V_1 to be uniformly distributed on a unit circle, according to the given ordering π_1. This defines Cartesian coordinates $(x(u), y(u)) \in \mathbb{R}^2$ for each $u \in V_1$. Then we compute for each $v \in V_2$ the *Cartesian barycenter*[3]

$$\mathrm{bary}(v) = \left(\frac{\sum_{u \in N^-(v)} x(u)}{|N^-(v)|}, \frac{\sum_{u \in N^-(v)} y(u)}{|N^-(v)|} \right)$$

of its predecessors $N^-(v)$ and sort the vertices circularly around the origin, i. e., by the angles of $\mathrm{bary}(v)$

[3]Note that the division by $|N^-(v)|$ can be omitted in an implementation, because it does not change the polar angle of $\mathrm{bary}(v)$.

in polar coordinates,

$$\beta(v) = \arctan \frac{y\left(\mathrm{bary}(v)\right)}{x\left(\mathrm{bary}(v)\right)} + \pi \cdot H\left(-x\left(\mathrm{bary}(v)\right)\right) \cdot \mathrm{sgn}\left(y\left(\mathrm{bary}(v)\right)\right),$$

where $H(x) = 0$ for $x \leq 0$ and $H(x) = 1$ for $x > 0$ is the unit step function. Many programming languages provide a specialized function `atan2(x, y)` for this purpose.

After sorting, we distribute the vertices of the second level uniformly on a concentric circle with radius 2 and choose for the offset of each edge one of -1, 0, or $+1$, whichever leads to the shortest edge in a geometric sense. Obviously, this algorithm has the same running time as its horizontal version:

Theorem 1: The running time of the Cartesian barycenter heuristic is $\mathcal{O}(|E| + |V| \log |V|)$.

C. Cartesian Median Heuristic

The Cartesian median heuristic is similar to the Cartesian barycenter heuristic. The only difference is that we take component-wise the x and y median instead of the component-wise barycenter. The running time stays the same, since $\mathrm{med}(v)$ can be computed in $\mathcal{O}(N^-(v))$, see [18]. The median values depend on the underlying coordinate system (origin and rotation). But since we use the same coordinates for all median computations, this is no problem. Rotated coordinate systems, however, might lead to different results.

D. Radial Sifting Heuristic

As a contrast to the fast and simple algorithms described above, we also developed an extension of the sifting heuristic, which is slower but generates fewer crossings. Sifting was originally introduced as a heuristic for vertex minimization in ordered binary decision diagrams [19] and later adapted for the (horizontal) one-sided crossing minimization problem [20]. The idea is to keep track of the objective function while moving a vertex $v \in V_2$ along a fixed ordering of all other vertices in V_2. Then v is placed to its locally optimal position. The method is thus an extension of the greedy-switch heuristic [21], where neighbors are only swapped if this does not increase the number of crossings. For crossing reduction the objective function is the number of crossings between the edges incident to the vertex under consideration and all other edges. Consecutively testing each vertex $v \in V_2$ on each position once is called a *sifting round.*

The efficient computation of crossing numbers in sifting for horizontal embeddings is based on the *crossing matrix.* Its entries correspond to the number of crossings caused by pairs of vertices in a particular relative ordering and can be computed as a preprocessing step. Whenever a vertex is placed to a new position, only a small number of updates is necessary. For radial embeddings, however, the crossings matrix cannot be computed in advance, because two vertices cannot be said to be in a particular (linear) relative order on radial levels.

Let $\mathcal{E} = (\pi, \psi)$ and $\mathcal{E}' = (\pi', \psi)$ be two embeddings of G, where $\mathcal{E}'$ is computed from $\mathcal{E}$ by swapping the vertex $v \in V_2$ and its successor $w \in V_2$ according to π_2, i. e., $\pi'_2(w) = \pi_2(v)$ and $\pi'_2(v) = \pi_2(v) + 1$. Since swapping positions of v and w only affects crossings of incident edges, the number of crossings in $\mathcal{E}'$ is efficiently computed as

$$\chi(\mathcal{E}') = \chi(\mathcal{E}) - c_{\mathcal{E}}(v, w) + c_{\mathcal{E}'}(v, w), \text{ where}$$
$$c_{\mathcal{E}}(v, w) = \sum_{u \in N^-(v)} \sum_{x \in N^-(w)} \chi_{\mathcal{E}}\left((u, v), (x, w)\right) .$$

Unfortunately, we cannot directly transfer the ideas of [4] for the efficient computation of that formula, because in radial sifting the crossing numbers also depend on the edge offsets, which are not constant in our approach. A change in the offset of an edge may affect all other edges. Therefore, the overall running time of this part of the algorithm for one sifting round is $\mathcal{O}(|E|^2)$ instead of $\mathcal{O}(|V||E|)$. The total running time of the algorithm, however, is dominated by the next step, anyway.

In addition to the position of vertex v, we also have to compute the offsets of the incident edges. As v moves along the second level circle in counterclockwise direction, we update the offsets accordingly. Because of Lemma 2 we do not consider each possible offset combination for each position of v. Intuitively, the parting of the edges should move around the first level circle in the same direction as v, but on the opposite side of the circle. Otherwise, the edges incident to v get longer and tend to increase the number of crossings. Thus, we only decrease edge offsets by 1, starting with $\psi(e) = 1$ for all incident edges e, and we also do this one by one in the order of the end vertices on level 1. The decision for which offsets are updated at which position of v is made subject to whether this leads to an improvement or not. Note that the parting may move around level 1 twice, as offsets are decreased from 1 to -1.

Algorithm 1 shows one round of radial sifting. We do not try the position $|V_2|-1$, because it is equivalent to position 0 modulo rotation.

Algorithm 1: RADIAL-SIFTING

```
   Input: Two level graph $G = (V_1 \dot\cup V_2, E, \phi)$ with
          radial embedding $\mathcal{E} = (\pi, \psi)$
   Output: Updated embedding $\mathcal{E}$, i. e., positions $\pi_2$
           and offsets $\psi$
1  foreach $v \in V_2$ with $\deg(v) > 0$ do
2      // put v to first position
3      foreach $w \in V_2$ with $\pi_2(w) < \pi_2(v)$ do
4          $\pi_2(w) \leftarrow \pi_2(w) + 1$
5      $\pi_2(v) \leftarrow 0$
6      let $\{v = v_0, \dots, v_{|V_2|-1}\} \leftarrow V_2$ be ordered by $\pi_2$
7      let $E_v \leftarrow \{e_0, \dots, e_{\deg(v)-1}\}$ be the edges
           $(u, v) \in E$ ordered by $\pi_1(u)$
8      // init offsets as 1
9      foreach $e_v \in E_v$ do $\psi(e_v) \leftarrow 1$
10     // init counters for pos, offset, parting, #cross.
11     $i^* \leftarrow 0$; $j^* \leftarrow j \leftarrow 0$; $l^* \leftarrow l \leftarrow 0$; $c^* \leftarrow c \leftarrow 0$
12     // search best position for v
13     for $i \leftarrow 0$ to $|V_2| - 2$ do
14         repeat
15             // try to improve part. by red. next offset
16             $c_1 \leftarrow \sum_{e \in E} \chi_{\mathcal{E}}(e_l, e)$
17             $\psi(e_l) \leftarrow j$
18             $c_2 \leftarrow \sum_{e \in E} \chi_{\mathcal{E}}(e_l, e)$
19             // if successful, try again, else restore
20             if $c_2 \le c_1$ then
21                 $c \leftarrow c - c_1 + c_2$
22                 $l \leftarrow l + 1$
23                 if $l = \deg(v)$ then
24                     $j \leftarrow j - 1$
25                     $l \leftarrow 0$
26             else $\psi(e_l) \leftarrow j + 1$
27         until $c_1 < c_2$
28         // store best pos, offset, parting, #cross.
29         if $c < c^*$ then $i^* \leftarrow i$; $j^* \leftarrow j$; $l^* \leftarrow l$; $c^* \leftarrow c$
30         // swap v and $v_{i+1}$ and update #cross.
31         let $E_{v_{i+1}}$ be the set of edges $(\cdot, v_{i+1}) \in E$
               incident to $v_{i+1}$
32         $c \leftarrow c - \sum_{e_v \in E_v} \sum_{e_{v_{i+1}} \in E_{v_{i+1}}} \chi_{\mathcal{E}}(e_v, e_{v_{i+1}})$
33         $\pi_2(v_{i+1}) \leftarrow i$; $\pi_2(v) \leftarrow i + 1$
34         $c \leftarrow c + \sum_{e_v \in E_v} \sum_{e_{v_{i+1}} \in E_{v_{i+1}}} \chi_{\mathcal{E}}(e_v, e_{v_{i+1}})$
35     // place v to best position
36     foreach $w \in V_2$ with $\pi_2(w) \ge i^*$ do
37         $\pi_2(w) \leftarrow \pi_2(w) + 1$
38     $\pi_2(v) \leftarrow i^*$
39     // set best offsets for v's incident edges
40     for $i \leftarrow 0$ to $l^* - 1$ do $\psi(e_i) \leftarrow j^*$
41     for $i \leftarrow l^*$ to $\deg(v) - 1$ do $\psi(e_i) \leftarrow j^* + 1$
```

Theorem 2: Given a 2-level graph $G = (V, E, \phi)$, the algorithm RADIAL-SIFTING runs in $\mathcal{O}(|V|^2 \cdot |E|)$ time.

Proof: For each node $v \in V_2$ the content of the repeat-until loop in lines 14–27 is executed $\mathcal{O}(|V| + \deg(v))$ times: once per position, and additionally once per shifted parting. It is thus executed $\mathcal{O}(|V|^2 + |E|)$ times in total. As the running times of lines 16 and 18 are $\mathcal{O}(|E|)$, the repeat-until loop contributes $\mathcal{O}(|V|^2 \cdot |E| + |E|^2)$ to the overall running time.

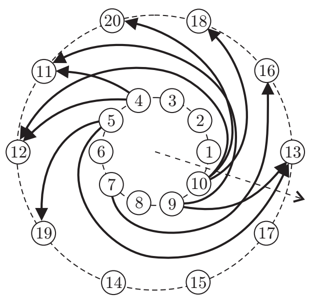
The only other relevant part are lines 32 and 34, which are executed once for each pair (v, v_{i+1}). Since the summation needs $\mathcal{O}(\deg(v) \cdot \deg(v_{i+1}))$, the total running time of this part is $\mathcal{O}(|E|^2)$ and is therefore dominated by the above. ■

To allow a harmonic drawing of the computed embedding in the next phase a final postprocessing which rotates level 2 with respect to uniform edge lengths is useful, e. g., see Fig. 5. Since our algorithm starts with an offset of 1 for every edge and stops at the first best parting among several others which are as good, a straightforward drawing of the embedding is twisted too much (in counter-clockwise direction). Thus the sum of the absolute edge lengths can be reduced by rotating level 2 with Algorithm 2: While assuming to have a drawing with uniform vertex distribution on both levels, we compute the average angle spanned by the edges and rotate the whole level 2 by this amount. However, this $\mathcal{O}(|E|)$ time postprocessing is only for aesthetic reasons and does neither affect the number of crossings nor the asymptotic running time.

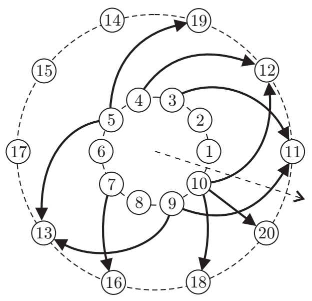
E. Experimental Results

To analyse the performance of our heuristics, we have implemented them in Java. Further, we have realized the corresponding horizontal versions to compare the resulting number of crossings with the radial algorithms. We have tested the implementations using a total number of 5000 random graphs: 50 graphs for each combination of the following parameters: $|V_1| = |V_2| \in \{20, 40, 60, 80, 100\}$ and $|E|/|V_2| \in \{1, \dots, 20\}$.

The experimental results in the Appendix show that all radial heuristics generate fewer crossings than their horizontal equivalents, experimentally by a factor of 0.7. This is a very encouraging result, since the running times of the radial algorithms (except sifting) are similar, see Fig. 10. Like in the horizontal case [22], Cartesian barycenter on average leaves slightly fewer crossings than Cartesian median. Another similarity is that radial sifting is the best among all three radial heuristics, but also the slowest. Usually only few



(a) Twisted embedding



(b) Figure 5(a) clockwise rotated by 4

Fig. 5. Postprocessing to reduce absolute edge lengths

Algorithm 2: RADIAL-SIFTING-POSTPROCESS

```
Input: Radial embedding $\mathcal{E} = (\pi, \psi)$ of a two level
       graph $G = (V_1 \dot\cup V_2, E, \phi)$
Output: Updated embedding $\mathcal{E}$, i. e., positions $\pi_2$
        and offsets $\psi$
1  // init vertex distances and average angle
2  $\epsilon_1 \leftarrow \frac{2\pi}{|V_1|}$; $\epsilon_2 \leftarrow \frac{2\pi}{|V_2|}$; $\delta \leftarrow 0$
3  // compute average angle spanned by edges
4  foreach $e = (u, v) \in E$ do
5      $\alpha_u \leftarrow \pi_1(u) \cdot \epsilon_1$; $\alpha_v \leftarrow \pi_2(v) \cdot \epsilon_2$
6      $\delta \leftarrow \delta + (\alpha_u - \alpha_v) + 2\pi \cdot \psi(e)$
7  $\delta \leftarrow \frac{\delta}{|E|}$
8  // if necessary, rotate level 2
9  $r \leftarrow \lfloor \frac{\delta}{\epsilon_2} + \frac{1}{2} \rfloor$
10 if $r < 0$ then
11     for $i \leftarrow 1$ to $|r|$ do
12         Counter-clockwise rotate level 2
13 else if $r > 0$ then
14     for $i \leftarrow 1$ to $r$ do
15         Clockwise rotate level 2
```

sifting rounds ($3-5$ for reasonable problem instances) are necessary to reach a local optimum for all vertices simultaneously, and the largest reduction of crossings usually occurs in the first round. In our experiments we further observed that the quality of radial sifting does not depend much on the quality of the initial embedding. However, a poor initialization increases the number of sifting rounds needed and thus raises the absolute running time.

V. Radial Coordinate Assignment

As already mentioned, in radial level drawings we draw the edge segments as segments of a spiral, unless they are radially aligned, in which case they are drawn as straight lines. This results in strictly monotone curves from inner to outer levels and ensures that segments do not cross inner level lines or unnecessarily each other. This phase is usually constrained not to change the vertex orderings computed previously, what is especially useful if the input embedding is a planar embedding, e. g., as generated by [3]. Further, the drawing algorithm should support commonly accepted criteria for readability and aesthetics, like small area, good separation of (dummy) vertices within a level, length and slope of edges, straightness of long edges, and balancing of edges incident to the same vertex. In our opinion edge bends in radial level drawings tend to be even more disturbing than in horizontal level drawings. Thus we base our algorithm on the approach of Brandes/Köpf [23] which guarantees at most two bends per edge. Further it prioritizes vertical alignment, which helps us to obtain radial alignment. The criterion of small area in horizontal coordinate assignment, i. e., to obtain small width, turns to uniform distribution of the vertices on the radial levels. As a consequence, a user parameter δ like in Sect. V-A is not needed. Since the input embedding for this phase maintains the position $\pi(v)$ for every vertex $v \in V \cup B$, the position of the ray is implicitly evident, i. e., on each level it lies between the two vertices with extremal positions.

A. Horizontal Coordinate Assignment

There are several algorithms for horizontal coordinate assignment [1], [24]–[31] using different approaches for the optimization of various objective functions or iterative improvement techniques. Most interesting is the Brandes/Köpf algorithm [23], which generates at most two bends per edge and draws every inner segment vertically if no two inner segments cross. Further it minimizes the horizontal stretch of segments and also gives good results for the other aesthetic criteria. The algorithm has $\mathcal{O}(N)$ running time and is fast in practice. For level planar embeddings Eades et al. [32] presented an algorithm that does not generate bends at all. However, it may need exponential area.

Since the horizontal drawing algorithm of Brandes/Köpf [23] is the basis of our radial drawing algorithm, we give an extended overview. It consists of three basic steps: vertical alignment, horizontal compaction, and balancing. The first two steps are carried out four times. After that, the four results are combined in the balancing step.

1) Vertical Alignment: The objective is to consecutively align each vertex with its left upper, right upper, left lower, and right lower median neighbor. Here and in Sect. V-A.2 we describe the alignment to the left upper median, the other three passes are analogous.

At the beginning all segments are removed from the graph which do not lead to an upper median neighbor, i. e., only candidates for vertical alignment are left, see Fig. 6(b). Then two alignments are conflicting if their corresponding edge segments cross or share a vertex. These conflicts are classified according to the number of involved inner edge segments. *Type* 2 *conflicts*, two crossing inner segments, are assumed to have been avoided by the crossing reduction phase and not to occur. For example this is automatically ensured by the barycenter and median methods. For sifting the absence of type 2 conflicts can be ensured by weighting each inner segment crossing with $|E|$ instead of 1. *Type* 1 *conflicts*, a non-inner segment crossing an inner segment, are resolved in favor of the inner segment. That means, the non-inner segment is removed from the graph. Finally, *type* 0 *conflicts*, two crossing non-inner segments, are resolved greedily in a leftmost fashion. That means, the right segment is removed from the graph. At this point there are no crossings left, see Fig. 6(c).

2) Horizontal Compaction: In the second step each maximum set of vertically aligned vertices, i. e., each connected component, is combined into a *block*, see Fig. 6(d). Consider the *block graph* obtained by introducing directed edges between each vertex and its successor (if any) on its level and by contracting the blocks into single vertices, see Fig. 6(e). A "horizontal" longest path layering on the block graph determines the x-coordinate of each block and thus of each contained vertex. Thereby the given minimum separation of the vertices δ is preserved.

The longest path layering leaves horizontal gaps between the blocks. Thus a further horizontal compaction is possible: The block graph with expanded blocks is partitioned into *classes*, see Fig. 6(f). The first class is defined as the set of vertices which are reachable from the top left vertex. Then the class is removed from the block graph. This is repeated, until every vertex is in a class. Within the classes the graph is already compact. Now the algorithm places the classes as close as possible except for minimum separation δ. In Fig. 6(f) this already happened. Fig. 6(g) shows the complete left upper layout.

3) Balancing: At this point we have four x-coordinates for each vertex. The two left (right) aligned assignments are shifted horizontally so that their minimum (maximum) coordinate agrees with the minimum (maximum) coordinate of the smallest width layout. The resulting coordinate is the average median[4] of the four intermediate coordinates. After reinserting all removed segments, the resulting drawing is obtained, see Fig. 6(h).

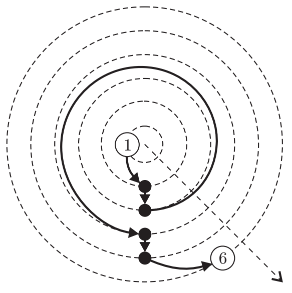


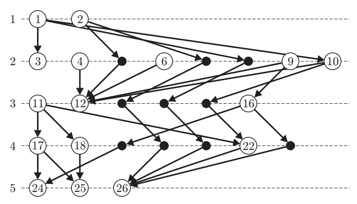
Fig. 7. Type 3 conflict

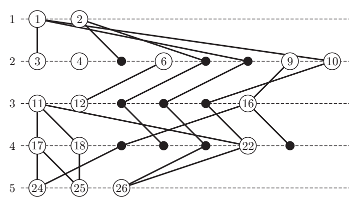
B. Preprocessing

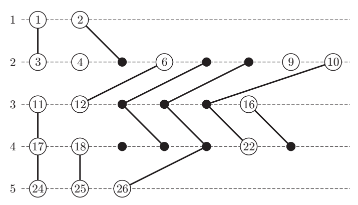
If an inner segment is a cut segment, i. e., if it crosses the ray, then the maximum of two bends for the corresponding long edge cannot be guaranteed, see Fig. 7 for an example. We call this situation a *type* 3 *conflict*. A simple solution is to demand the absence of inner cut segments in the input embedding, similar as it is done with type 2 conflicts. A different, more constructive and always doable approach described in the following, is to eliminate the conflicts by changing the position of the ray. This strategy changes the offset of some edges and thus changes the embedding. But this does not affect a later drawing.

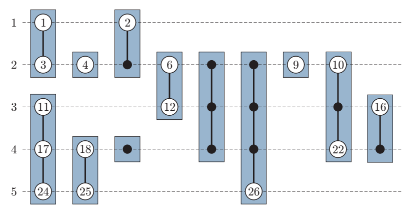
Before we continue with the description of the elimination algorithm, we discuss an important property of radial level embeddings:

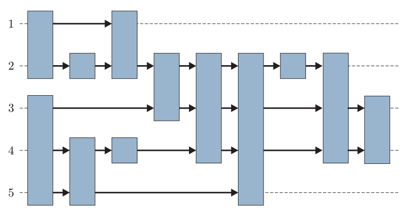
[4]If the median is not unique, the average median is defined as the average of the two median values.

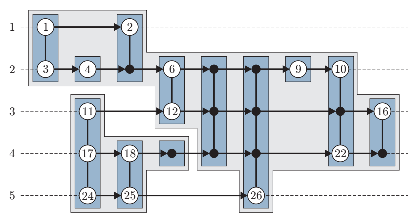

(a) Level embedding

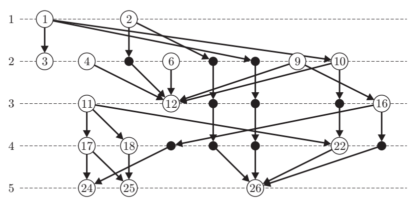

(b) Candidates


(c) Alignment


(d) Blocks


(e) Block graph


(f) Classes


(g) Left upper layout

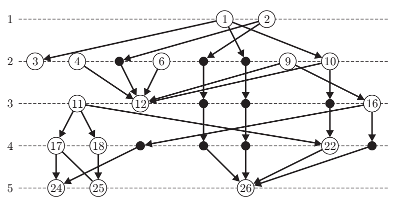

(h) Final balanced layout

Fig. 6. Stages of the Brandes/Köpf algorithm

Lemma 4: Let $E'_i = \{ (u, v) \mid u \in B_{i-1}, v \in B_i \} \subseteq E$ be the set of all inner segments between levels $i-1$ and i with $2 < i < k$. Then for any two edges $e_1, e_2 \in E'_i : |\psi(e_1) - \psi(e_2)| \leq 1$.

Proof: In the extreme case let $e_1, e_2 \in E'_i$ for $2 < i < k$ be inner segments with $\psi(e_1) = \max\{\psi(e) \mid e \in E'_i\}$ and $\psi(e_2) = \min\{\psi(e) \mid e \in E'_i\}$. Now assume that $\psi(e_1) > \psi(e_2) + 1$. As a consequence e_1 and e_2 cross. This is a type 2 conflict and contradicts the absence of type 2 conflicts in the input embedding. ■

In a first step to eliminate type 3 conflicts we consecutively *unwind* the levels in ascending order from 3 to $k-1$ with Algorithm 3. Between levels 1 and 2 resp. $k-1$ and k there are no inner segments. Clearly, level i is unwound by rotating the whole outer graph, i. e., all levels $\geq i$ are rotated by multiples of 360 degrees. Please note that UNWIND-LEVEL updates only offsets of edges between levels $i-1$ and i. The position of the ray, i. e., the ordering of the vertices, remains the same.

Lemma 5: After unwinding for each inner segment $e \in E$: $\psi(e) \in \{0, +1\}$.

Proof: Lemma 4 implies for each inner segment $e = (u, v)$ with $\phi(v) = i$ that $\psi(e) \leq 1$. Additionally

Algorithm 3: UNWIND-LEVEL

```
Input: Radial embedding $\mathcal{E} = (\pi, \psi)$ of a level graph
       $G = (V \cup B, E, \phi)$ and level $i$ with $2 < i < k$
Output: Updated embedding $\mathcal{E}$, i. e., offsets $\psi$ of
        inner segments entering level $i$
1 $m \leftarrow \min\{\psi(e) \mid e = (u, v) \in E, u \in B_{i-1}, v \in B_i\}$
2 foreach segment $e = (u, v) \in E$ with $v \in V_i \cup B_i$ do
3     $\psi(e) \leftarrow \psi(e) - m$
```

$\psi(e)$ cannot be negative because we have subtracted the minimum over all inner segments entering level i. Since this argument holds for every level $2 < i < k$, the claim follows. ■

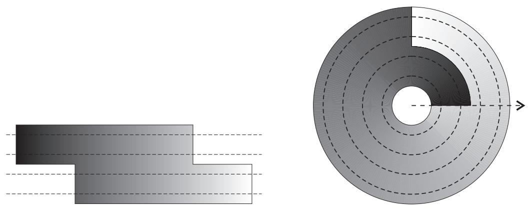
Lemma 6: After unwinding there are no two dummy vertices $v, v' \in B_i$ on the same level i with $\psi\big((u, v)\big) = 0$, $\psi\big((u', v')\big) = +1$, and $v \prec v'$ for any $u, u' \in B_{i-1}$.

Proof: Follows directly from the absence of type 2 conflicts. ■

Now we use rotation as described in Sect. IV to eliminate the remaining crossings of inner segments with the ray. Please note that rotation of a single level i is different from rotating levels during unwinding. Here we do not rotate by (multiples of) 360 degrees in general and do not rotate all levels $\geq i$ simultaneously. Let $B'_i \subseteq B_i$ be the set of dummy vertices incident to an incoming inner segment $e = (u, v)$ with $\psi(e) = +1$. Let $v = \operatorname{argmax}\{\pi(v) \mid v \in B'_i\}$. We rotate level i clockwise until the ray enters the position after v, i. e., until v is the last vertex on i and thus $v = \operatorname{argmax}\{\pi(v) \mid v \in V_i \cup B_i\}$. We use the clockwise direction, because according to Lemma 6 we do not generate new type 3 conflicts this way. Finally, all inner segments have an offset of 0. The overall running time is $\mathcal{O}(N)$.

C. Intermediate Horizontal Layout

In the next step we generate a horizontal layout of the radial level embedding with the Brandes/Köpf algorithm. Therefore we ignore all cut segments. Remember that the embedding is free of type 3 conflicts. Thus all inner segments of an edge are aligned vertically. The resulting layout will later be transformed into a concentric layout by concentrically connecting the ends of the horizontal level lines with their beginnings. Therefore, we must take into account that circumferences of radial level lines grow with ascending level numbers. Thus we use a minimum vertex separation distance $\delta_i = \frac{1}{i}$ for each horizontal level i, which is in each case indirectly proportional to i.



(a) Horizontal (b) Radial

Fig. 8. Overlap of the left and right contour

In this way we achieve a uniform minimum arc length between two neighbor vertices on every radial level line with the radial transformation described in the next section, since we use the level numbers $1, 2, \ldots, k$ as radii.

D. Radial Layout

At this stage every vertex $v \in V$ has Cartesian coordinates $\big(x(v), y(v) = \phi(v)\big) \in \mathbb{R} \times \mathbb{R}$. For the transformation into a radial drawing we interpret these coordinates as polar coordinates and transform them with Eq. (1) into Cartesian coordinates $\big(x_r(v), y_r(v)\big) \in \mathbb{R} \times \mathbb{R}$. The position of the ray denotes 0 degrees.

$$\begin{aligned}\big(x_r(v), y_r(v)\big) &= \\ &\Big(y(v) \cdot \cos\left(\tfrac{2\pi}{z} \cdot x(v)\right), y(v) \cdot \sin\left(\tfrac{2\pi}{z} \cdot x(v)\right)\Big)\end{aligned} \quad (1)$$

The factor $\frac{2\pi}{z}$ normalizes the length of the horizontal level lines to the circumferences of the radial level lines. We set $z = \max\big\{\max\{x(v') \mid v' \in V_i \cup B_i\} - \min\{x(v') \mid v' \in V_i \cup B_i\} + \delta_i \mid 1 \leq i \leq k\big\}$, i. e., z is the largest horizontal distance between two vertices on the same level i plus δ_i. The addend δ_i is necessary to maintain the minimum distance between the first and the last vertex, since they become neighbors on the radial level line. Let i_z be the level which defines z. The normalization automatically realizes the necessary overlap between the left and the right contour of the horizontal layout when drawn radially, see Fig. 8. Level i_z is the widest level and thus i_z defines the maximum overlap of the contours.

After drawing the vertices, we draw the edges as segments of a spiral. Each point p of a straight line segment $e = (u, v)$ is defined by Eq. (2) for $0 \leq t \leq 1$.

$$\big(x(p), y(p)\big) = (1-t)\big(x(u), y(u)\big) + t\big(x(v), y(v)\big) \quad (2)$$

The coordinates of p can be transformed with Eq. (1). But e can be a cut segment, which winds multiple times clockwise or counter-clockwise around the center. Therefore we rather use Eq. (3) which simulates this behavior horizontally. Imagine $|\psi(e)|+1$ copies of the layout placed in a row, cf. Fig. 9. If $\psi(e) \geq 0$, then imagine e drawn as straight line from u in the leftmost layout to v in the rightmost layout. Otherwise, draw e from u in the rightmost layout to v in the leftmost one. Any two neighboring layouts of the row are separated by δ_{i_z}, i. e., the x-coordinate of the leftmost vertex in the right layout is the x-coordinate of the rightmost vertex of the left layout plus δ_{i_z}.

$$\begin{aligned}\big(x(p), y(p)\big) =&(1-t)\big(x(u), y(u)\big)\\ &+ t\big(x(v) + \psi(e)\cdot z, y(v)\big)\end{aligned} \tag{3}$$

For all edges with offset 0 there is only one possible direction without crossing the ray, i. e., there is only one copy in the row. Equation (3) inserted in Eq. (1) for drawing a spiral segment between u and v results in the following equation:

$$\begin{aligned}&\big(x_r(p), y_r(p)\big) = \big((1-t)y(u) + t\cdot y(v)\big)\\ &\cdot\Bigg(\cos\left(\tfrac{2\pi}{z}\cdot\big((1-t)x(u) + t\cdot(x(v)+\psi(e)\cdot z)\big)\right),\\ &\quad \sin\left(\tfrac{2\pi}{z}\cdot\big((1-t)x(u) + t\cdot(x(v)+\psi(e)\cdot z)\big)\right)\Bigg)\end{aligned} \tag{4}$$

If $t = 0.5$, then p lies on a concentric circle with radius $\frac{\phi(u)+\phi(v)}{2}$, because the radius of the spiral segment grows proportional to the concentric distance between p and $\phi(u)$. To obtain smooth edges, the number of supporting points $s\colon E \to \mathbb{N}$ needed for drawing edges $e = (u, v)$ with an approximating polyline or spline depends on the edge length and a quality factor $Q \geq 1$.

$$\begin{aligned}s(e) &\sim \phi(v)\cdot\left(\left|\tfrac{2\pi}{z}\cdot x(v) - \tfrac{2\pi}{z}\cdot x(u) + \psi(e)\cdot 2\pi\right|\right)\cdot Q\\ &\sim \phi(v)\cdot\left(\left|\tfrac{x(v)-x(u)}{z} + \psi(e)\right|\right)\cdot Q\end{aligned} \tag{5}$$

In the special case of $|V_1| = 1$, e. g., in Fig. 7, it is more aesthetic pleasing to place $v \in V_1$ into the concentric center, cf. Fig. 1(b). Thus we renumber the levels by $\phi'(w) = \phi(w) - 1$ for all $w \in V \cup B - \{v\}$, set $x_r(v) = y_r(v) = 0$, layout $G' = (V \cup B - \{v\}, E - \{\,(v, w) \mid w \in V\,\}, \phi')$, and draw each edge (v, w) as a straight line. To get a readable picture in the case $|V_1| > 1$, Eades [10] suggests to set the diameter of the first level to the radial distance between the radial level lines. To achieve this with our algorithm, we use $0.5, 1.5, 2.5, \ldots, k-1.5, k-0.5$ as level numbers/radii. An equivalent solution is to double the number of levels $k' = 2k$, to renumber the levels by $\phi'(v) = 2\phi(v) - 1$ for all $v \in V \cup B$, to generate a drawing of $G' = (V \cup B, E, \phi')$, and finally to zoom by a factor of $\frac{1}{2}$.

Usually we draw on a canvas which has dimensions $a \times b$ and has the origin in the upper left corner. Thus for each vertex or supporting point p we do the following: With the translation $\big(x_r(p), y_r(p)\big) = \big(x_r(p) + \frac{a}{2}, y_r(p) + \frac{b}{2}\big)$ we move the origin to the center. In order to use the entire drawing space, we scale the layout by $\big(x(p), y(p)\big) = \big(x(p), y(p)\big)\cdot\frac{\min\{a,b\}}{2k}$.

Since the elimination of type 3 conflicts generates no new crossings and Eqs. (1) and 4 are bijective we do not change the crossing number given by the embedding. A radial level planar embedding is drawn planar. Adopting the common assumption that drawing a line (here an edge as a spiral segment with its supporting points) needs $\mathcal{O}(1)$ time, we obtain an $\mathcal{O}(N)$ running time.

VI. Conclusion

We extended three well known crossing reduction techniques to radial level drawings. In practice, all algorithms are fast enough to be applied to reasonably large graphs. We showed by empirical evidence, that using radial instead of horizontal level lines reduces the number of crossings significantly. Further we have presented a new linear time algorithm for drawing level graphs (assigning coordinates) in a radial fashion. To check its performance and to visually confirm the good quality of the resulting drawings we realized a prototype as a plug-in for Gravisto [33] in Java. For a given embedding, the coordinates of a graph with $N = 50{,}000$ can be computed in less than 50 seconds on a 1.8 GHz PC with 768 MB RAM. For computing radial embeddings of graphs of this size sifting is too slow and one should choose the faster but qualitatively inferior barycenter or median method, analogously to the recommendation for horizontal embeddings.

Future research can address a more efficient sifting algorithm. Also, there are some interesting problems which we do not touch in this paper: Can the number of crossings $\chi(\mathcal{E})$ in a radial embedding $\mathcal{E}$ be computed in $\mathcal{O}(\chi(\mathcal{E}))$ time? Is $\chi(\mathcal{E}) \leq 3\chi(G)$ (or similar) for an embedding $\mathcal{E}$ computed by one-sided Cartesian median heuristic on a 2-level graph G as it is for horizontal median [17]? Are there efficient radial

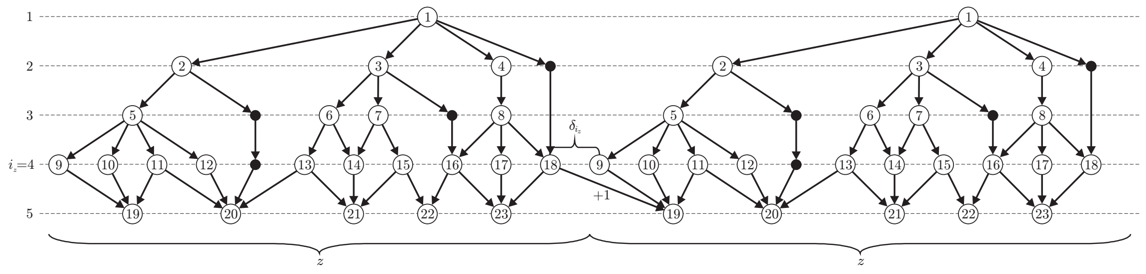


Fig. 9. Simulation of cut edges

extensions of other crossing reduction heuristics? A radial crossing reduction algorithm that already avoids type 3 conflicts in this phase would be helpful, since our elimination approach may create many crossings of non-inner segments with the ray.

Acknowledgment

The author would like to thank Michael Forster for contributing to the parent papers [15], [34] and Matthias Höllmüller for expunging some bugs and implementing our algorithms in Gravisto [33] within a network analysis package.

References

[1] K. Sugiyama, S. Tagawa, and M. Toda, "Methods for visual understanding of hierarchical system structures," *IEEE Trans. Syst., Man, Cybern.*, vol. 11, no. 2, pp. 109–125, 1981.

[2] M. Kaufmann and D. Wagner, *Drawing Graphs*, ser. LNCS. Springer, 2001, vol. 2025.

[3] C. Bachmaier, F. J. Brandenburg, and M. Forster, "Radial level planarity testing and embedding in linear time," *Journal of Graph Algorithms and Applications*, vol. 9, no. 1, pp. 53–97, 2005.

[4] M. Baur and U. Brandes, "Crossing reduction in circular layout," in *Proc. Workshop on Graph-Theoretic Concepts in Computer Science, WG 2004*, ser. LNCS, J. Hromkovic, M. Nagl, and B. Westfechtel, Eds., vol. 3353. Springer, 2005, pp. 332–343.

[5] E. Mäkinen, "On circular layouts," *International Journal of Computer Mathematics*, vol. 24, pp. 29–37, 1988.

[6] J. M. Six and I. G. Tollis, "A framework and algorithms for circular drawings of graphs," *Journal of Discrete Algorithms*, vol. 4, no. 1, pp. 25–50, 2006.

[7] E. Di Giacomo, W. Didimo, L. Giuseppe, and H. Meijer, "Computing radial drawings on the minimum number of circles," *Journal of Graph Algorithms and Applications*, vol. 9, no. 3, pp. 347–364, 2005.

[8] M. Carpano, "Automatic display of hierarchized graphs for computeraided decision analysis," *IEEE Trans. Syst., Man, Cybern.*, vol. 10, no. 11, pp. 705–715, 1980.

[9] M. G. Reggiani and F. E. Marchetti, "A proposed method for representing hierarchies," *IEEE Trans. Syst., Man, Cybern.*, vol. 18, no. 1, pp. 2–8, 1988.

[10] P. Eades, "Drawing free trees," *Bulletin of the Institute of Combinatorics and its Applications*, vol. 5, pp. 10–36, 1992.

[11] U. Brandes, P. Kenis, and D. Wagner, "Centrality in policy network drawings," in *Proc. Graph Drawing, GD 1999*, ser. LNCS, J. Kratochvíl, Ed., vol. 1731. Springer, 1999, pp. 250–258.

[12] ——, "Communicating centrality in policy network drawings," *IEEE Trans. Visual. Comput. Graphics*, vol. 9, no. 2, pp. 241–253, 2003.

[13] T. J. Jankun-Kelly and K.-L. Ma, "Moiregraphs: Radial focus+context visualization and interaction for graphs with visual nodes," in *Proc. IEEE Symposium on Information Visualization, INFOVIS 2003*, T. Munzner and S. North, Eds. IEEE Computer Society Press, 2003, pp. 59–66.

[14] K.-P. Yee, D. Fisher, R. Dhamija, and M. Hearst, "Animated exploration of dynamic graphs with radial layout," in *Proc. IEEE Symposium on Information Visualization, INFOVIS 2001*, 2001, pp. 43–50.

[15] C. Bachmaier, F. Fischer, and M. Forster, "Radial coordinate assignment for level graphs," in *Proc. Computing and Combinatorics, COCOON 2005*, ser. LNCS, L. Wang, Ed., vol. 3595. Springer, 2005, pp. 401–410.

[16] E. G. Coffman and R. L. Graham, "Optimal scheduling for two processor systems," *Acta Informatica*, vol. 1, pp. 200–213, 1972.

[17] P. Eades and N. C. Wormald, "Edge crossings in drawings of bipartite graphs," *Algorithmica*, vol. 11, no. 1, pp. 379–403, 1994.

[18] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms.* MIT Press, 2000.

[19] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," in *Proc. IEEE/ACM International Conference on Computer Aided Design, ICCAD 1993.* IEEE Computer Society Press, 1993, pp. 42–47.

[20] C. Matuszewski, R. Schönfeld, and P. Molitor, "Using sifting for k-layer straightline crossing minimization," in *Proc. Graph Drawing, GD 1999*, ser. LNCS, J. Kratochvíl, Ed., vol. 1731. Springer, 1999, pp. 217–224.

[21] P. Eades and D. Kelly, "Heuristics for reducing crossings in 2-layered networks," *Ars Combinatorica*, vol. 21, no. A, pp. 89–98, 1986.

[22] M. Jünger and P. Mutzel, "2-layer straightline crossing minimization: Performance of exact and heuristic algorithms," *Journal of Graph Algorithms and Applications*, vol. 1, no. 1, pp. 1–25, 1997.

[23] U. Brandes and B. Köpf, "Fast and simple horizontal coordinate assignment," in *Proc. Graph Drawing, GD 2001*, ser. LNCS, P. Mutzel, M. Jünger, and S. Leipert, Eds., vol. 2265. Springer, 2001, pp. 31–44.

[24] C. Buchheim, M. Jünger, and S. Leipert, "A fast layout algorithm for k-level graphs," in *Proc. Graph Drawing, GD*

2000, ser. LNCS, J. Marks, Ed., vol. 1984. Springer, 2001, pp. 229–240.

[25] P. Eades, X. Lin, and R. Tamassia, "An algorithm for drawing hierarchical graphs," *International Journal of Computational Geometry & Applications*, vol. 6, pp. 145–156, 1996.

[26] P. Eades and K. Sugiyama, "How to draw a directed graph," *Journal of Information Processing*, vol. 13, no. 4, pp. 424–437, 1990.

[27] E. R. Gansner, E. Koutsofios, S. North, and K.-P. Vo, "A technique for drawing directed graphs," *IEEE Transactions on Software Engineering*, vol. 19, no. 3, pp. 214–230, 1993.

[28] E. R. Gansner, S. C. North, and K.-P. Vo, "DAG, a program that draws directed graphs," *Software – Practice and Experience*, vol. 17, no. 1, pp. 1047–1062, 1988.

[29] G. Sander, "Graph layout through the VCG tool," in *Proc. Graph Drawing, GD 1994*, ser. LNCS, R. Tamassia and I. G. Tollis, Eds., vol. 894. Springer, 1995, pp. 194–205.

[30] ——, "A fast heuristic for hierarchical Manhattan Layout," in *Proc. Graph Drawing, GD 1995*, ser. LNCS, F. J. Brandenburg, Ed., vol. 1027. Springer, 1996, pp. 447–458.

[31] ——, "Graph layout for applications in compiler construction," *Theoretical Computer Science*, vol. 217, pp. 175–214, 1999.

[32] P. Eades, Q.-W. Feng, and X. Lin, "Straight-line drawing algorithms for hierarchical graphs and clustered graphs," in *Proc. Graph Drawing, GD 1996*, ser. LNCS, S. C. North, Ed., vol. 1190. Springer, 1997, pp. 113–128.

[33] "Gravisto. Graph Visualization Toolkit," http://gravisto.fmi.uni-passau.de/, University of Passau.

[34] C. Bachmaier and M. Forster, "A radial adaption of the Sugiyama framework for hierarchical graph drawing," University of Passau, Tech. Rep. MIP-0603, April 2006.

Appendix
Crossing Reduction Benchmark Results

Figure 10 provides benchmarks comparing horizontal barycenter (HB), horizontal median (HM), and horizontal sifting (HS) crossing reduction heuristics with their radial variants, i. e., Cartesian barycenter (CB), Cartesian median (CM), and radial sifting (RS). The diagrams show that radial sifting is the best algorithm leaving dramatically fewer crossings than the others, but it is also the slowest, and that radial barycenter is the fastest. The same facts hold for the corresponding horizontal algorithms, which is folklore.

Christian Bachmaier received a diploma and a Doctorate in Computer Science from the University of Passau in 2000 and 2004 respectively. He now has a postdoctoral appointment at the University of Passau in the group Prof. Dr. Brandenburg. His research interests include algorithm engineering, computional complexity, data structures, efficient graph algorithms, graph drawing for the visualization of information, and graph planarity.

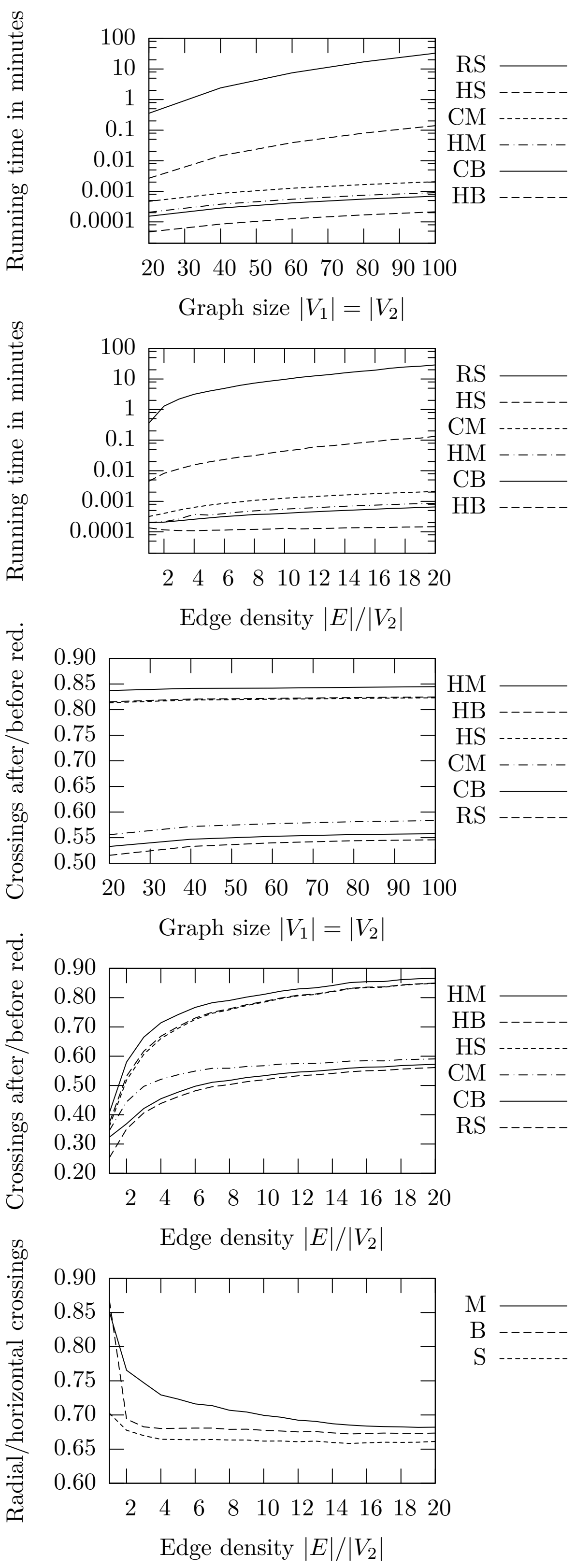


Fig. 10. Benchmarks

A.3 Articles for Sect. 4

At last [5] and [3] are listed in this order.

Cyclic Leveling of Directed Graphs

Christian Bachmaier, Franz J. Brandenburg,
Wolfgang Brunner, and Gergö Lovász

University of Passau, Germany
{bachmaier|brandenb|brunner|lovasz}@fim.uni-passau.de

Abstract. The Sugiyama framework is the most commonly used concept for visualizing directed graphs. It draws them in a hierarchical way and operates in four phases: cycle removal, leveling, crossing reduction, and coordinate assignment.
However, there are situations where cycles must be displayed as such, e. g., distinguished cycles in the biosciences and processes that repeat in a daily or weekly turn. This forbids the removal of cycles. In their seminal paper Sugiyama et al. also introduced recurrent hierarchies as a concept to draw graphs with cycles. However, this concept has not received much attention since then.
In this paper we investigate the leveling problem for cyclic graphs. We show that minimizing the sum of the length of all edges is $\mathcal{NP}$-hard for a given number of levels and present three different heuristics for the leveling problem. This sharply contrasts the situation in the hierarchical style of drawing directed graphs, where this problem is solvable in polynomial time.

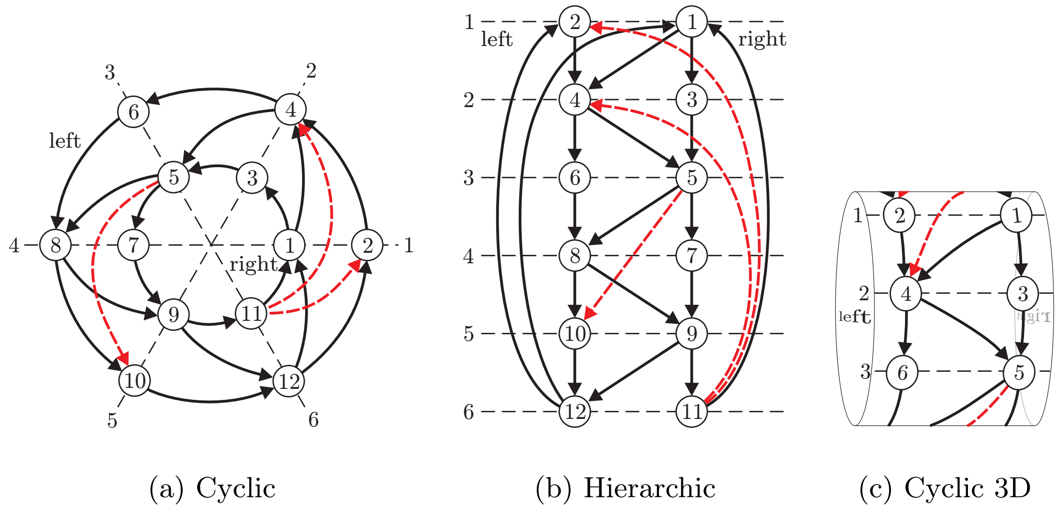
1 Introduction

The Sugiyama framework [8] is among the most intensively investigated algorithms in graph drawing. It is the standard technique to draw directed graphs, and displays them in an hierarchical manner. This is well-suited particularly for directed acyclic graphs, which are drawn top-down (or left to right) and level by level. These drawings reflect the underlying graph as a partial order. Typical applications are schedules, UML diagrams and flow charts.

In the general case, the Sugiyama framework first destroys cycles. In the decycling phase it removes or redirects some edges until the resulting graph is acyclic. However, there are many situations, where this procedure is inacceptable. For example, there are well-known cycles in the biosciences, and it is a common standard there to display these cycles as such. These cycles often serve as a landmark [7]. Another application for cycles are repeating processes, such as daily, weekly or monthly schedules with almost the same tasks. Here again it is important that these cycles are clearly visible in a “nice” drawing.

In their original paper from 1981 [8], Sugiyama et al. have proposed a solution for both the hierarchical and the cyclic style. The latter is called *recurrent hierarchy*. A recurrent hierarchy is a level graph with additional edges from the last to the first level. Here, two drawings are natural: The first is a 2D drawing, where the levels are rays from a common center, and are sorted counterclockwise by their number, see Fig. 1(a). All nodes of one level are placed at different positions on their ray and an edge $e = (u, v)$ is drawn as a monotone counterclockwise curve from u to v wrapping around the center at most once. The second is a 3D drawing on a cylinder, see Fig. 1(c). A combination of the two drawing methods would be the best of both worlds: An interactive 2D view which shows horizontal levels. This view can be scrolled upwards and downwards infinitely and always shows a different part of the cylinder, e. g., the front view of Fig. 1(c).

Recurrent hierarchies are known to most graph drawers – but unnoticed. A planar recurrent hierarchy is shown on the cover of the book by Kaufmann and Wagner [6]. There it is stated that recurrent hierarchies are "unfortunately [. . .] still not well studied". The reason is that they are much harder. Intuitively, there is no start and no end, there are no top and bottom levels. Formally, we pinpoint a problem which is tractable in the hierarchical style and is intractable in the cyclic style.



(a) Cyclic (b) Hierarchic (c) Cyclic 3D

Fig. 1. Example drawings

In cyclic drawings edges are irreversible and cycles are represented in a direct way. Thus, the cycle removal phase disappears from the common Sugiyama framework. This saves much effort, since the underlying problem is the $\mathcal{NP}$-hard feedback arc set problem [5]. Another advantage are short edges. The sum of the edge length can be smaller than in the hierarchical case: Consider a cycle consisting of three nodes. The only way to draw this graph in the Sugiyama framework is to reverse one edge which will then span two levels. Therefore, the sum of the edge length will be four. In the cyclic case this graph can be drawn on three levels s. t. each edge has span one. Moreover, the cyclic style reduces the number of crossings in general. See Fig. 1(a) and (b) as an example. At the threshold with no crossings [2], there are cyclic level planar graphs which are not level planar. Here, consider Fig. 1 with the solid edges only.

Note that any Sugiyama drawing is a cyclic Sugiyama drawing which discards the option to draw edges between the last and first level. Therefore, all benefits of such drawings exist in the cyclic case as well. However, the sum of the edge length and the number of crossings will often be smaller.

In this paper we consider the leveling phase for the cyclic Sugiyama framework. In the hierarchical version this phase is generally solved by topological sorting, or more advanced, by the Coffman-Graham algorithm [3, 6]. As our main result we show that minimizing the sum of the length of all edges is $\mathcal{NP}$-hard for a given number of levels. This sharply contrasts the hierarchical case. Then we introduce three heuristics for the cyclic leveling problem, and evaluate them experimentally within the Gravisto system [1].

2 Preliminaries

Let $G = (V, E)$ be a directed graph. For a given $k \in \mathbb{N}$ we call $\phi\colon V \to \{1, 2, \dots, k\}$ a level assignment and $G = (V, E, \phi)$ a cyclic k-level graph. We denote with $\deg(v)$ the degree of a node $v \in V$. For two nodes $u, v \in V$ let $\text{span}(u, v) = \phi(v) - \phi(u)$ if $\phi(u) < \phi(v)$, and $\text{span}(u, v) = \phi(v) - \phi(u) + k$ otherwise. For an edge $e = (u, v) \in E$ we define $\text{span}(e) = \text{span}(u, v)$ and $\text{span}(G) = \sum_{e \in E} \text{span}(e)$. For a set of edges $E' \subseteq E$ we define $\text{span}(E') = \sum_{e \in E'} \text{span}(e)$. $\text{next}(l) = (l \mod k) + 1$ denotes the level after l. For a node $v \in V$ and a subset $V' \subset V$ we set $E(v, V') = \{\, (u, v) \in E \mid u \in V' \,\} \cup \{\, (v, w) \in E \mid w \in V' \,\}$.

3 Complexity of Cyclic Leveling

In this section we consider different leveling problems and compare their complexity in the hierarchical and the cyclic style. The graphs $G = (V, E)$ are directed in both cases and are acyclic in the hierarchical case.

Definition 1 (Height and Width). *Let G be a directed graph which is drawn s. t. the edges connect vertices on different levels and are uni-directed from the start level to a successive level. Let the* height *be the number of levels and let the* width *be the maximal number of nodes on a level.*

We can now state our leveling problems, both for the common hierarchical style and for the cyclic style of recurrent hierarchies.

Problem 1. Let $k \in \mathbb{N}$. Does there exist a leveling of G with height at most k?

Problem 2. Let $\omega \in \mathbb{N}$. Does there exist a leveling of G with width at most ω?

Problem 3. Let $k, \omega \in \mathbb{N}$. Does there exist a leveling of G with height at most k and width at most ω?

In the hierarchical case problems 1 and 2 are easy: The former can be solved in linear time by the longest path search algorithm [6], whereas, the latter is trivial as each graph has a leveling with width 1 by placing each node on its own level according to a topological sorting of G. Problem 3 is $\mathcal{NP}$-hard as this corresponds to precedence constrained scheduling [5].

In the cyclic case all these problems are easy. Note that an edge $e = (u, v)$ does not impose any constraint on the leveling of the nodes u and v. u can have a smaller level, a larger level, and even the same level as v. Therefore, the answer to Problems 1 and 2 is yes (if $k, \omega > 0$). For Problem 3 there is a cyclic leveling if $|V| \leq k \cdot \omega$ by arbitrarily placing vertices in a $k \times \omega$ grid.

Problem 4. Let $l \in \mathbb{N}$. Does there exist a leveling of G with $\operatorname{span}(G) \leq l$?

In the hierarchical case minimizing the span can be formulated as an ILP:

$$\min \sum_{(u,v)\in E} (\phi(v) - \phi(u)) \tag{1}$$

$$\forall v \in V : \phi(v) \in \mathbb{N} \tag{2}$$

$$\forall e = (u, v) \in E : \phi(v) - \phi(u) \geq 1 \tag{3}$$

This ILP can be solved in polynomial time, since the constraint matrix is totally unimodular [6]. Therefore, Problem 4 has a polynomial time complexity in the hierarchical case as well. In the cyclic case the span can no longer be formulated by a system of linear equations, as a case differentiation or the modulo operation is needed.

As a degenerated case we may place all nodes on a single level. Then all edges have span 1 which is obviously minimal. Therefore, we sharpen Problem 4:

Problem 5. Let $l, k \in \mathbb{N}$. Does there exist a leveling of G with exactly k levels with $\operatorname{span}(G) \leq l$?

Problem 5 is simple for $k = 1$, as such a leveling exists if $l \geq |E|$. For $k > 1$ we now show that the problem is $\mathcal{NP}$-hard. We use two different reductions for $k = 2$ and $k > 2$. For $k = 2$ we use the $\mathcal{NP}$-hard bipartite subgraph problem [5]:

Problem 6 (Bipartite subgraph). Let $G = (V, E)$ be an undirected graph and $k \in \mathbb{N}$. Does there exist a bipartite subgraph G' of G with at least k edges?

Lemma 1. *Let $G = (V, E)$ be an undirected graph and $l \in \mathbb{N}$. Let $G^* = (V, E^*)$ be a directed version of G with an arbitrary direction for each edge. G contains a bipartite subgraph G' with at least l edges if and only if there exists a leveling of G^* on two levels with* $\operatorname{span}(G^*) \leq 2|E| - l$.

Proof. "$\Rightarrow$": Let $G' = (V', E')$ be a bipartite subgraph of G with at least l edges. Let $V_1 \dot{\cup} V_2 = V'$ be the partition of the node set with all edges of E' between V_1 and V_2. We construct the following leveling for G^*: For each node $v \in V_1$ we set $\phi(v) = 1$, for each node $v \in V_2$ we set $\phi(v) = 2$, and for all nodes $v \in V \setminus (V_1 \cup V_2)$

we set $\phi(v) = 1$. Then each edge in E' has span 1 and all other edges have span 1 or 2. Thus, $\text{span}(G^*) \leq |E'| + 2(|E| - |E'|) = 2|E| - E' \leq 2|E| - l$.

"$\Leftarrow$": Let ϕ be a leveling of G^* with $\text{span}(G^*) \leq 2|E| - l$. Let V_1 and V_2 be the nodes of V on level 1 and 2, respectively. Let $E' \subseteq E$ be the set of edges e s. t. one end node is in V_1 and the other is in V_2. Then, $G' = (V, E')$ is bipartite. All edges in E' have span 1 in the leveling ϕ and all other edges have span 2. As $\text{span}(G^*) \leq 2|E| - l = 1 \cdot l + 2(|E| - l)$, there are at least l edges with span 1. As these edges are in E', G' is a bipartite subgraph of G with at least l edges. □

For $k > 2$ we use graph k-colorability, which is $\mathcal{NP}$-hard for a fixed $k > 2$ [5]:

Problem 7 (Graph k-colorability). Let $G = (V, E)$ be an undirected graph and let $k \in \mathbb{N}$. Does there exist a coloring $c : V \to \{1, \dots, k\}$, s. t. $c(u) \neq c(v)$ for every edge $e = \{u, v\} \in E$?

Lemma 2. *Let $G = (V, E)$ be an undirected graph and let $k \in \mathbb{N}$. Let $G' = (V, E')$ with E' containing the edges (u, v) and (v, u) for each edge $\{u, v\} \in E$. G is k-colorable if and only if G' has a leveling on k levels with* $\text{span}(G') \leq k \cdot |E|$.

Proof. Let $e = \{u, v\} \in E$. Note that for each leveling ϕ of G' and each edge $e = (u, v) \in E'$ the sum of the spans of (u, v) and (v, u) is either k (if $\phi(u) \neq \phi(v)$) or $2k$ (if $\phi(u) = \phi(v)$). Thus, $\text{span}(G') \geq k \cdot \frac{|E'|}{2} = k \cdot |E|$.

"$\Rightarrow$": Let c be a coloring of G. Set $\phi = c$. Then, for each edge with end nodes u and v in G (and G') $\phi(u) \neq \phi(v)$ holds. Thus, each pair of edges (u, v) and (v, u) in sum has span k and $\text{span}(G') = k \cdot |E|$ holds.

"$\Leftarrow$": Let ϕ be a leveling of G' with $\text{span}(G') \leq k \cdot |E|$. Then, $\text{span}(G') = k \cdot |E|$ and for each edge $(u, v) \in E'$ $\phi(u) \neq \phi(v)$ holds. Thus, $c = \phi$ is a correct coloring. □

Theorem 1. *Let $G = (V, E)$ be a directed graph and $l, k \in \mathbb{N}$ ($k \geq 2$). The problem whether there exists a leveling of G on k levels with* $\text{span}(G) \leq l$ *is $\mathcal{NP}$-complete.*

Proof. Lemma 1 and Lemma 2 show that the problem is $\mathcal{NP}$-hard for $k = 2$ and $k > 2$, respectively. The problem is obviously in $\mathcal{NP}$. □

4 Heuristics

As minimizing the span of a graph in a cyclic leveling with k levels is $\mathcal{NP}$-complete, we have to use heuristics. Known approaches from the hierarchical case as the longest path method [6] or the Coffman-Graham algorithm [3] cannot be easily adapted to the cyclic case. They heavily rely on the fact that the graph is acyclic and start the leveling process at nodes with no incoming edges. As it is not guaranteed that such nodes exist in the cyclic case at all, we introduce three new heuristics. They are evaluated experimentally in Sect. 5.

The input to the algorithms are the number of levels k and the maximum number of nodes on a level ω. The output is the leveling $\phi : V \to \{1, \dots, k\}$. The parameter k is either given by the user or it is pre-computed, e. g., as the average length of simple cycles detected by a depth first search of the graph.

Table 1. Complexity of leveling (k as height and ω as width)

	hierarchical	cyclic
Minimizing k	$\mathcal{O}(\lvert V\rvert + \lvert E\rvert)$, by longest path	Set $\phi : V \to \{1\}$
Minimizing ω	$\mathcal{O}(\lvert V\rvert + \lvert E\rvert)$, by ϕ = topsort	Choose injective ϕ
Leveling with k and ω given	$\mathcal{NP}$-hard, precedence constrained scheduling	Test $k \cdot \omega \geq \lvert V\rvert$
Minimizing k with ω given	$\mathcal{NP}$-hard for arbitrary $\omega > 2$	Set $k = \lceil \frac{\lvert V\rvert}{\omega} \rceil$
Minimizing ω with k given	$\mathcal{NP}$-hard for $k > 2$	Set $\omega = \lceil \frac{\lvert V\rvert}{k} \rceil$
Minimizing span(G) with k given	$\mathcal{P}$, by LP	$\mathcal{NP}$-hard for $k > 1$

4.1 Breadth First Search

The breadth first search (BFS) heuristic (Algorithm 1) is rather simple: We choose an arbitrary start node v, set $\phi(v) = 1$ and perform a directed BFS from v. When we reach a node w for the first time using an edge (u, w), we set $\phi(w) = \text{next}(\phi(u))$ if this level does not contain ω nodes already. Otherwise, we move w to the first non-full level.

Using this heuristic the tree edges will have a rather short span. But the back edges are not taken into account for the leveling at all. Thus, these edges can be arbitrarily long.

Lemma 3. *The BFS leveling heuristic needs* $\mathcal{O}(|V| + |E| + k^2)$ *time.*

Proof. BFS runs in $\mathcal{O}(|V| + |E|)$ time. In addition we must keep and update an array N of size k. $N[i]$ denotes the first non-full level from level i. At most all k levels can get full which costs $\mathcal{O}(k)$ time for each. □

4.2 Minimum Spanning Tree

This heuristic has similarities to the algorithm of Prim [4], which computes the minimum spanning tree (MST) of a graph. We sequentially level the nodes by a greedy algorithm. Let $V' \subset V$ be the set of already leveled nodes. When we level a node v, all edges in $E(v, V')$ get a fixed span. Therefore, we set $\phi(v)$ s. t. span$(E(v, V'))$ is minimized. Note that there are possibly more edges incident to v which are also incident to not yet leveled nodes. These edges will be considered when the second end node is leveled.

We decide in which order to add the nodes by using a distance function $\delta(v)$. We discuss four options:

Algorithm 1: breadthFirstSearchLeveling

```
Input: $G$: a directed graph, $k$: the number of levels,
       $\omega$: the maximum number of nodes on each level
Output: $\phi$: a cyclic leveling of $G$
1  Queue $Q \leftarrow \emptyset$
2  Leveling $\phi \leftarrow \emptyset$
3  foreach $u \in V$ do $u.marked \leftarrow false$
4  foreach $l \in \{1, \dots, k\}$ do $N[l] \leftarrow l$
5  foreach $u \in V$ do
6      if $\neg u.marked$ then
7          $Q.append(u)$
8          $u.marked \leftarrow true$
9          $\phi(u) \leftarrow N[1]$
10         $updateN(N[1])$
11         while $\neg Q.isEmpty()$ do
12             $v \leftarrow Q.removeFirst()$
13             foreach neighbor $w$ of $v$ do
14                 if $\neg w.marked$ then
15                     $w.marked \leftarrow true$
16                     $\phi(w) \leftarrow N[\text{next}(\phi(v))]$
17                     $updateN(\phi(w))$
18                     $Q.append(w)$
19 return $\phi$
```

Minimum Increase in Span (MST_MIN) We choose the node which will create the minimum increase in span in the already leveled graph:

$$\delta_{\text{MIN}}(v) = \min_{\phi(v) \in \{1,\dots,k\}} \text{span}(E(v, V')) \tag{4}$$

Minimum Average Increase in Span (MST_MIN_AVG) Using the distance function δ_{MIN} will place nodes with a low degree first, as nodes with a higher degree will almost always cause a higher increase in span. Therefore, considering the increase in span per edge is reasonable:

$$\delta_{\text{MIN\_AVG}}(v) = \min_{\phi(v) \in \{1,\dots,k\}} \frac{\text{span}(E(v, V'))}{|E(v, V')|} \tag{5}$$

We distribute isolated nodes evenly on the non-full levels in the end.

Maximum (Average) Increase in Span (MST_MAX(_AVG)) Choose the node which causes the maximum (average) increase in span per edge:

$$\delta_{\text{MAX}}(v) = \frac{1}{\delta_{\text{MIN}}(v)}, \qquad \delta_{\text{MAX\_AVG}}(v) = \frac{1}{\delta_{\text{MIN\_AVG}}(v)} \tag{6}$$

The idea behind this is the following: A node which causes a high increase in span will cause this increase when leveled later as well. But if we level this node now, we can possibly level other adjacent, not yet leveled nodes in a better way.

Note that we only use the distance function $\delta(v)$ to determine which node to level next. When we level a node v, we set $\phi(v)$ s. t. the increase in span will be minimized. In some cases several levels for v will create the same increase in span. We will then choose a level for v which minimizes $\sum_{e\in E(v,V')} \text{span}(e)^2$ as well. Thus, we assign v a level which is more centered between its leveled adjacent nodes. In each case we can only use a level which has not yet ω nodes on it. Nodes with already leveled neighbors block a place on their optimal level s. t. they can later be placed on the level. Algorithm 2 shows the complete heuristic.

Algorithm 2: minimumSpanningTreeLeveling

```
Input: $G$: a directed graph, $k$: the number of levels,
       $\omega$: the maximum number of nodes on each level
Output: $\phi$: a cyclic leveling of $G$
1  Heap $H \leftarrow \emptyset$
2  Leveling $\phi \leftarrow \emptyset$
3  foreach $u \in V$ do
4      $u.status \leftarrow white$
5      $\delta(u) \leftarrow \infty$
6  foreach $u \in V$ do
7      if $u.status = white$ then
8          $\delta(u) \leftarrow 0$
9          $H.insert(u)$
10         while $\neg H.isempty()$ do
11             $v \leftarrow H.removeMin()$
12             $v.status \leftarrow black$
13             $\phi(v) \leftarrow getOptimalLevel(v)$
14             foreach neighbor $w$ of $v$ with $w.status \neq black$ do
15                 $\delta(w) \leftarrow computeDistance(w)$
16                 $\phi(w) \leftarrow getOptimalLevel(w)$
17                 if $w.status = gray$ then
18                     $H.update(w)$
19                 else
20                     $w.status \leftarrow gray$
21                     $H.insert(w)$
22 return $\phi$
```

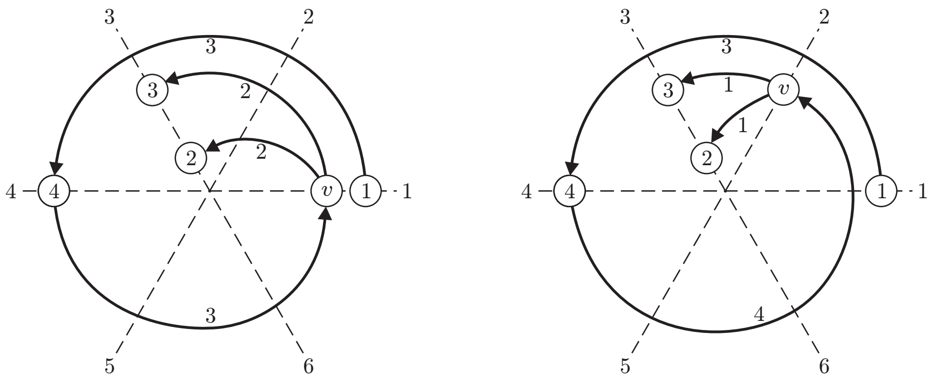
Lemma 4. *The MST heuristic needs $\mathcal{O}(|V|\log|V| + k \cdot \deg(G) \cdot |E|)$ time.*

Proof. The time complexity is dominated by the while loop. Here, removing each node from the heap costs $\mathcal{O}(|V|\log|V|)$. Each edge $e = (w, z) \in E$ may change its span whenever a neighbor v of w (or z) is fixed on a level. In this case each of the k levels is tested for w (or z). Thus, we get $\mathcal{O}(k \cdot (\deg(w) + \deg(z)))$ for e and $\mathcal{O}(k \cdot \deg(G) \cdot |E|)$ for all edges. Finally, updating all neighbors in the heap costs $\mathcal{O}(|E| \log |V|)$ (or $\mathcal{O}(|E|)$ using a Fibonacci heap). $\square$

4.3 Force Based

Spring embedders use a physical model to simulate the edges as springs [6]. Forces between nodes are computed and the nodes are moved accordingly. Transferring this idea to the cyclic leveling problem, we could use a force function similar to conventional energy based placement algorithms as follows:

$$\text{force}(v) = \sum_{(v,w)\in E} (\text{span}(v,w) - 1)^2 - \sum_{(u,v)\in E} (\text{span}(u,v) - 1)^2 \tag{7}$$



(a) v in energy minimum of (7) (b) v in energy minimum of (8)

Fig. 2. Force based placement of node v

However, moving a node to its energy minimum using this force will not minimize the span of the graph, i. e., (7) minimizes the deviation between the edge lengths, e. g., see Fig. 2. Furthermore, the span may increase when moving a node towards its energy minimum, as some edges can flip from span 1 to span k. We solve this problem by using directly the span as the (undirected) force which is minimized:

$$\text{force}(v) = \text{span}(E(v,V)) \tag{8}$$

We move the node with the maximum impacting force. And we directly move the node to its energy minimum, which is the level s. t. the span is minimized. For this, we test all possible (non-full) levels. Note that moving all nodes at once would not decrease time complexity here. Algorithm 3 shows the pseudo code.

As an initial leveling we either use a random leveling (SE_RND) or the result of the best minimum spanning tree heuristic MST_MIN_AVG (SE_MST).

Algorithm 3: forceBasedLeveling

```
Input: $G$: a directed graph, $k$: the number of levels,
       $\omega$: the maximum number of nodes on each level
Output: $\phi$: a cyclic leveling of $G$
1  Heap $H \leftarrow \emptyset$
2  $\phi \leftarrow computeInitialLeveling()$
3  foreach $v \in V$ do
4      $computeForce(v)$
5  while $improvement \wedge iterations < limit$ do
6      foreach $v \in V$ do
7          $H.insert(v)$
8      while $\neg H.isEmpty()$ do
9          $v \leftarrow H.removeMax()$
10         $\phi(v) \leftarrow energyMinimalLevel(v)$
11         foreach $neighbor\ w\ of\ v$ do
12             $updateForce(w, v)$
13 return $\phi$
```

Lemma 5. *In the force based heuristic $\mathcal{O}(|V| \log |V| + k \cdot |E|)$ time is needed for each iteration.*

Proof. Inserting all nodes in the heap can be implemented in $\mathcal{O}(|V|)$ time. Removing each node from the heap has time complexity $\mathcal{O}(|V| \log |V|)$. Computing the energy minimal level for v costs $\mathcal{O}(k \cdot \deg(v))$, which is $\mathcal{O}(k \cdot |E|)$ for all nodes. Computing the new force is possible in time $\mathcal{O}(1)$ for each neighbor of v, in $\mathcal{O}(\deg(v))$ for all neighbors and $\mathcal{O}(|E|)$ in total. The $\mathcal{O}(|E|)$ updates in the heap cost $\mathcal{O}(|E| \log |V|)$ (or $\mathcal{O}(|E|)$ using a Fibonacci heap). □

5 Empirical Results

In this section we evaluate and compare the heuristics with each other and with an optimal leveling. The optimal leveling is computed by a branch and bound algorithm which can be used for graphs up to 18 nodes.

In Fig. 3 the running times of the algorithms are shown. Figure 4 compares the calculated spans of the heuristics with the optimal span. For a better pairwise comparison of the heuristics, Fig. 5 only shows their results.

For Fig. 3 and 5 the number of nodes $|V|$ was increased by steps of 50 each time. For each size 10 graphs with $|E| = 5|V|$ were created randomly. For Fig. 4 10 graphs for each size $|V|$ and $|E| = 2|V|$ were used. In all three diagrams k and ω were set to $\sqrt{2|V|}$, s. t. there were $2|V|$ possible node positions. Each heuristic was applied to each graph $\min(|V|, 30)$ times using different start nodes resp. initial levelings and choosing the average.

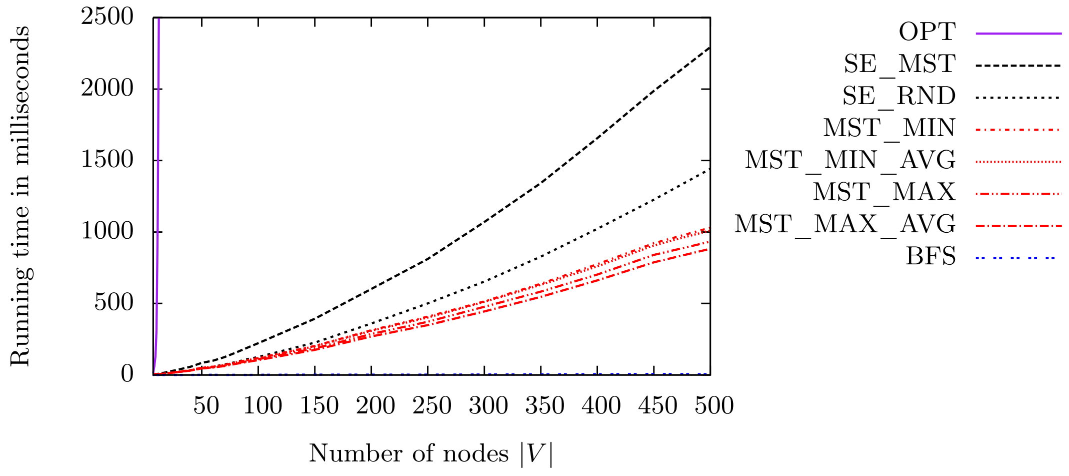


Fig. 3. Running times of the algorithms

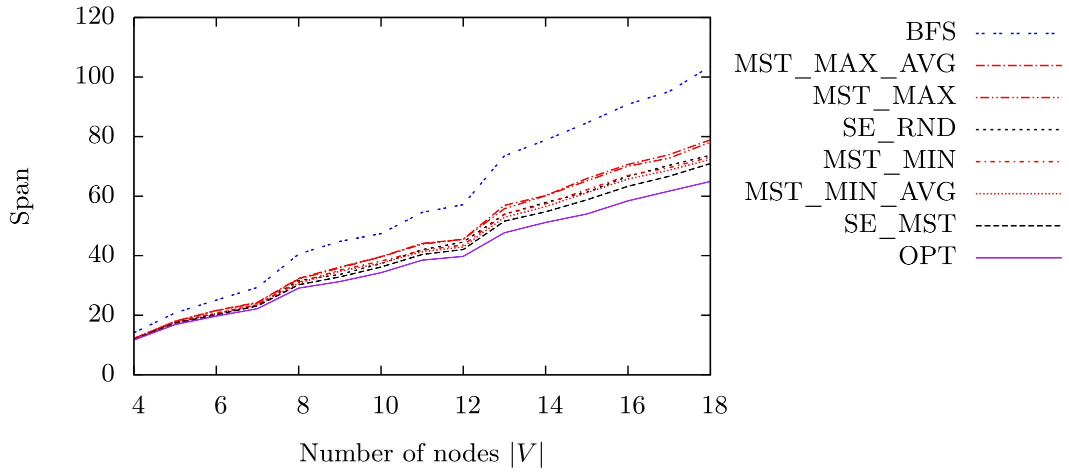


Fig. 4. Average spans of small graphs

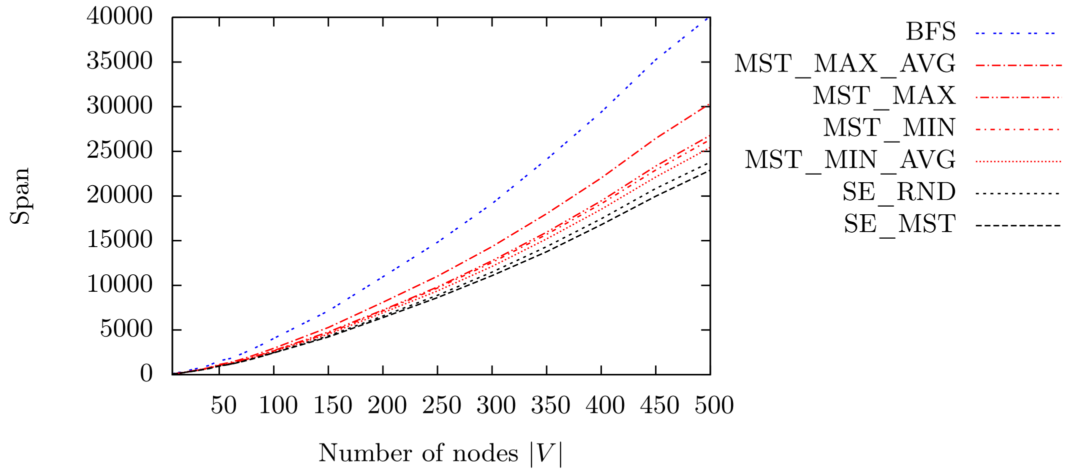


Fig. 5. Average spans of large graphs

The benchmarks show the practical performance of the algorithms. All tests were run on a 2.8 GHz Celeron PC under the Java 6.0 platform from Sun Microsystems, Inc. within the Gravisto framework [1].

As expected the force based heuristics with MST initialization computes the best leveling and the results are close to the optimum. All MST variants do not differ very much, but MST_MIN_AVG seems to be the best. The results can be improved by applying the heuristics i times to the same graph with i different start nodes or different initial levelings, respectively, and choosing the best result. However, the price is an i times higher running time.

6 Summary and Open Problems

The leveling problem has turned out to be essentially different in the hierarchical and cyclic style. We have shown different optimization goals for the cyclic leveling compared to the goals of the hierarchic leveling. For the reasonable minimization of the sum of the edge lengths we have shown the $\mathcal{NP}$-hardness and presented three practical heuristics for the problem.

Open problems are the approximation ratios of our heuristics, other quality measures for cyclic drawings, the best number of levels, and the completion of the cyclic style to a cyclic Sugiyama framework.

References

1. Bachmaier, C., Brandenburg, F.J., Forster, M., Holleis, P., Raitner, M.: Gravisto: Graph visualization toolkit. In: Pach, J. (ed.) GD 2004. LNCS, vol. 3383, pp. 502–503. Springer, Heidelberg (2005)
2. Bachmaier, C., Brunner, W.: Linear time planarity testing and embedding of strongly connected cyclic level graphs. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 136–147. Springer, Heidelberg (2008)
3. Coffman, E.G., Graham, R.L.: Optimal scheduling for two processor systems. Acta Informatica 1(3), 200–213 (1972)
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press, Cambridge, 2nd edn. (2001)
5. Garey, M.R., Johnson, D.S.: A Guide to the Theory of NP-Completeness. W. H. Freemann, New York (1979)
6. Kaufmann, M., Wagner, D.: Drawing Graphs, LNCS, vol. 2025. Springer, Heidelberg (2001)
7. Michal, G. (ed.): Biochemical Pathways: An Atlas of Biochemistry and Molecular Biology. Wiley, New York (1999)
8. 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)

Coordinate Assignment for Cyclic Level Graphs

Christian Bachmaier[1], Franz J. Brandenburg[1],
Wolfgang Brunner[1], and Raymund Fülöp[2]

[1] University of Passau, Germany,
{bachmaier|brandenb|brunner}@fim.uni-passau.de
[2] Technische Universität München, Germany, fueloep@in.tum.de

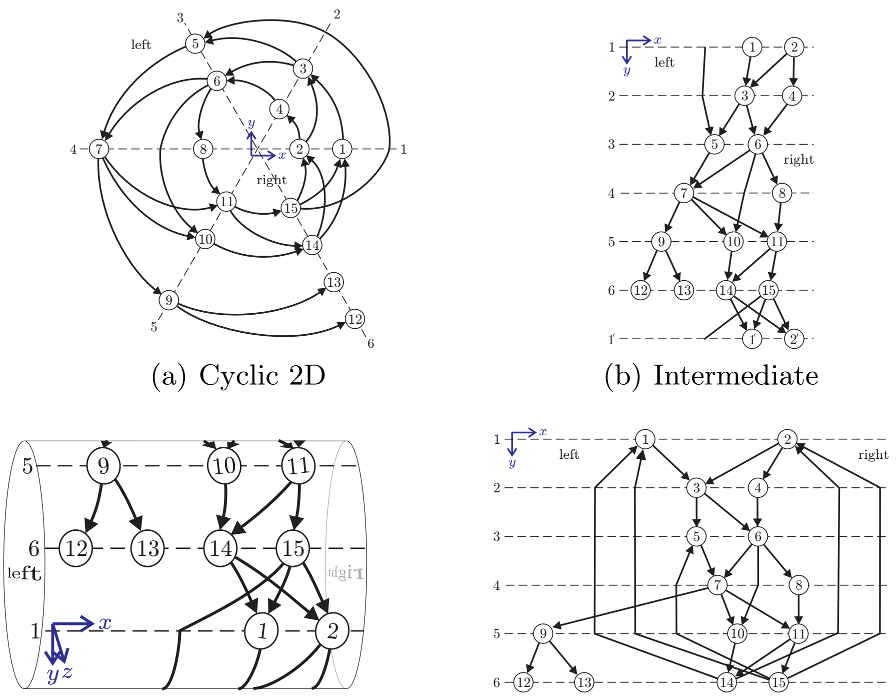
Abstract. The Sugiyama framework is the most commonly used concept for visualizing directed graphs. It draws them in a hierarchical way and operates in four phases: cycle removal, leveling, crossing reduction, and coordinate assignment. However, there are situations where cycles must be displayed as such, e. g., distinguished cycles in the biosciences and scheduling processes which repeat in a daily or weekly turn. This excludes the removal of cycles. In their seminal paper Sugiyama et al. introduced recurrent hierarchies as a concept to draw graphs with cycles. However, this concept has not received much attention in the following years. In this paper we supplement our cyclic Sugiyama framework and investigate the coordinate assignment phase. We provide an algorithm which runs in linear time and constructs drawings which have at most two bends per edge and use quadratic area.

1 Introduction

The Sugiyama framework [9] is among the most intensively studied algorithms in graph drawing. It is the standard technique to draw directed graphs, and displays them in a hierarchical manner. It consists of the four phases of cycle removal, leveling, crossing reduction, and coordinate assignment. Typical applications are schedules, UML diagrams, and flow charts.

In its first phase the Sugiyama framework destroys all cycles. However, there are many situations where this is unacceptable. There are well-known cycles in the biosciences [7], where it is a common standard to display these cycles as such. Another inevitable use are repeating processes, such as daily, weekly, or monthly schedules which define the Periodic Event Scheduling Problem [8].

In their seminal paper [9], Sugiyama et al. proposed a solution for both the hierarchic and the cyclic style. The latter is called a *recurrent hierarchy* which is a level graph with additional edges from the last to the first level. It can be drawn in 2D where the levels are rays from a common center (see Fig. 1(a)) and each edge $e = (u, v)$ is a monotone counterclockwise poly-spiral segment from u to v wrapping around the center at most once. An alternative is a 3D drawing on a cylinder (see Fig. 1(c)). A combination would be the best of both worlds: an interactive 2D view with horizontal levels. It can be scrolled upwards and downwards infinitely and always shows a different part of the cylinder, see Fig. 1(b) for a snap shot, which also represents our intermediate drawing.



(a) Cyclic 2D (b) Intermediate

(c) Cyclic 3D (d) Hierarchic

Fig. 1. Example drawings

In cyclic drawings edges are irreversible and cycles are represented in a direct way. Thus, the cycle removal phase is not needed. This saves much effort, since the underlying feedback arc set problem is $\mathcal{NP}$-hard [4]. Further advantages over hierarchic drawings (see Fig. 1(d)) are shorter edges and fewer crossings.

A planar recurrent hierarchy is shown on the cover of the textbook by Kaufmann and Wagner [6]. There it is stated that recurrent hierarchies are "unfortunately [...] still not well studied". After investigating the leveling phase [1], we consider the coordinate assignment phase for the cyclic case. There are several algorithms for non-cyclic coordinate assignment [6]. We modify the established algorithm of Brandes and Köpf [3] for cyclic level graphs and provide a linear time algorithm using quadratic area and with at most two bends per edge.

2 Preliminaries

A *cyclic k-level graph* $G = (V, E, \phi)$ $(k \geq 2)$ is a directed graph without self-loops with a given surjective level assignment of the vertices $\phi\colon V \to \{1, 2, \ldots, k\}$. Let $V_i \subset V$ be the set of vertices v with $\phi(v) = i$. For two vertices $u, v \in V$ let $\mathrm{span}(u, v) := \phi(v) - \phi(u)$ if $\phi(u) < \phi(v)$ and $\mathrm{span}(u, v) := \phi(v) - \phi(u) + k$ otherwise. For an edge $e = (a, b) \in E$ we define $\mathrm{span}(e) := \mathrm{span}(a, b)$. An edge e with $\mathrm{span}(e) = 1$ is *short*, otherwise *long*. A graph is *proper* if all edges are short. Each cyclic level graph can be made proper by adding $\mathrm{span}(e) - 1$ dummy vertices for each edge e and thus splitting e in $\mathrm{span}(e)$ many short edges, which we call the *segments* of e. In total, this leads up to $\mathcal{O}(|E| \cdot k)$ new vertices. The first and the last segment of each edge are its *outer segments*, and all other segments between two dummy vertices are its *inner segments*. A proper cyclic k-level graph $G = (V, E, \phi, <)$ is *ordered* if $<$ is a total ordering for each V_i $(1 \le i \le k)$. In accordance to [3] we say that in an ordered cyclic level graph there are two *conflicting* segments if they cross or share a vertex. Conflicts are of *type 0, 1* or *2*, if they are induced by 0, 1, or 2 inner segments, respectively.

We represent drawings of cyclic level graphs in an *intermediate drawing* in the remainder of the paper assigning each vertex v two coordinates $x(v) \in \mathbb{R}$ and $y(v) = \phi(v) \in \mathbb{N}$. The x-coordinate increases from *left* to *right*, the y-coordinate increases *downwards* in edge direction, see Fig. 1(b). All vertices on level 1 are duplicated on level $k+1$ using the same x-coordinates. Each segment $s = (u, v)$ is drawn straight-line from $\big(x(u), y(u)\big)$ to $\big(x(v), y(u)+1\big)$ with *slope* $\frac{1}{x(v)-x(u)}$. A *2D drawing* as in Fig. 1(a) is obtained from an intermediate drawing by transforming each point $p = (x(p), y(p))$ of the plane to $\big(x_{2\mathrm{D}}(p), y_{2\mathrm{D}}(p)\big) = \big(r(p) \cdot \cos(\alpha(p)), r(p) \cdot \sin(\alpha(p))\big)$, with the radius $r(p) = (\mathrm{offset}_x + \max_{v \in V}(x(v))) - x(p) \cdot \delta_x$ and the angle $\alpha(p) = (y(p) - 1) \cdot \frac{2\pi}{k}$. The constant offset_x defines the minimum distance of a vertex to the center and δ_x the minimum distance of vertices on the same level. A *3D drawing* as in Fig. 1(c) uses the coordinates $\big(x_{3\mathrm{D}}(p), y_{3\mathrm{D}}(p), z_{3\mathrm{D}}(p)\big) = \big(x(p) \cdot \delta_x, -r_k \cdot \sin(\alpha(p)), r_k \cdot \cos(\alpha(p))\big)$ where r_k is the radius of the cylinder. These equations transform straight lines of the intermediate drawing to spiral segments in the 2D or 3D drawings.

A drawing is *(cyclic level) plane* if the edges do not cross except on common endpoints. A cyclic k-level graph is *(cyclic level) planar* if such a drawing exists.

3 Layout Algorithm

In this section we describe our coordinate assignment phase for cyclic level graphs. We adapt the algorithm of Brandes and Köpf [3] and use their notation. Like them, we also assume that the crossing reduction has avoided type 2 conflicts. These can be avoided even for cyclic level graphs as shown recently [5].

The input to our algorithm is the output of the third phase and thus a proper ordered cyclic level graph. Note that dummy vertices were introduced after the leveling. Algorithm 1 consists of three basic steps: block building (lines 4–5), horizontal compaction (lines 6–12), and balancing (line 14) which reflect the steps in [3]. The first two steps are carried out four times (*runs*) for each combination of left/right with up/down alignment (line 2). The four results are merged by the balancing step. We describe the left top run only. The other three runs are realized by flipping the graph horizontally and/or vertically before and after (lines 3, 13) each run. The computed intermediate drawing can be transformed into the 2D or 3D drawing, where dummy vertices are replaced by edge bends.

In the cyclic case there may be unavoidable cyclic dependencies in the left-to-right ordering among vertically aligned paths. Thus, it is impossible to draw inner segments vertically, in general. We solve this problem by shearing the drawing of such a cycle s. t. all inner segments have the same slope.

Algorithm 1: cyclicCoordinateAssignment

Input: $G = (V, E, \phi, <)$: An ordered and proper cyclic k-level graph
Output: Coordinates $(x(v), y(v))$ for each $v \in V$ in the intermediate drawing $\mathcal{I}$

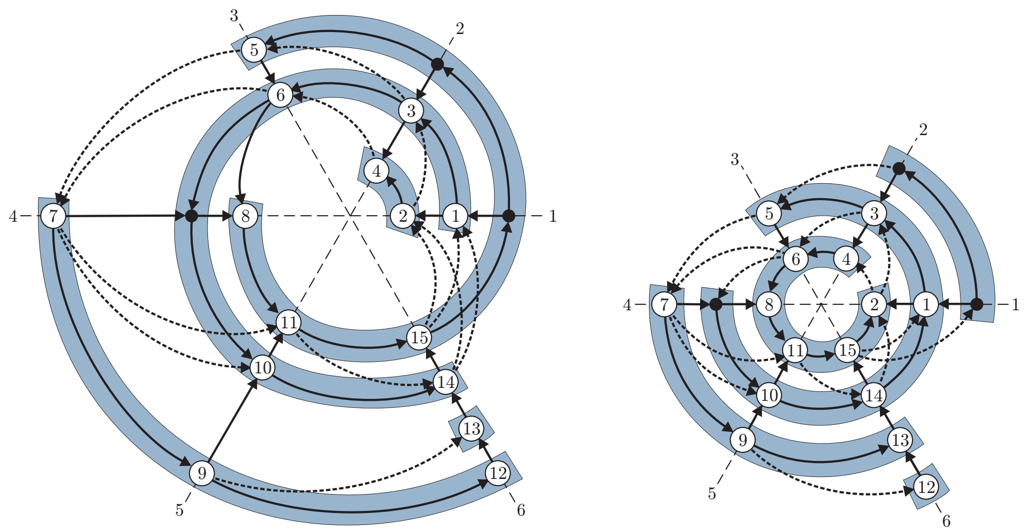
```
1  $\mathcal{P} \leftarrow \emptyset$
2  foreach $(h, v) \in \{left, right\} \times \{up, down\}$ do
3      $G' \leftarrow$ flip$(G, h, v)$                          // according to current run
4      $H \leftarrow$ buildCyclicBlockGraph$(G')$
5      splitLongBlocks$(H)$                          // split long and closed blocks
6      $\mathcal{S} \leftarrow$ computeSCCs$(H)$
7      foreach complex SCC $S \in \mathcal{S}$ do
8          $S' \leftarrow$ cutSCC$(S)$                    // returns non-cyclic block graph
9          width$(S') \leftarrow$ compact$(S')$            // using two topsorts
10         shear$(S', -(\text{wind}(S') \cdot k)/\text{width}(S'))$   // shear $S'$ with given slope
11         $\mathcal{S} \leftarrow \mathcal{S} \setminus S \cup S'$
12     compact$(\mathcal{S})$                              // globally all SCCs
13     $\mathcal{P} \leftarrow \mathcal{P} \cup$ flip$(\mathcal{S}, h, v)$
14 $\mathcal{I} \leftarrow$ balance$(\mathcal{P})$                      // balance four runs
15 return $\mathcal{I}$
```

3.1 Block Building

The block building phase is done in the same way as in [3]. We try to align vertices with its median adjacent vertices to blocks and remove all other segments level by level until we obtain a cyclic path graph and thus a cyclic block graph.

Definition 1. *A* cyclic path graph *$H' = (V, E_{intra}, \phi, <)$ is an ordered and proper cyclic level graph with a plane embedding respecting the ordering $<$. Each vertex of H' has indegree and outdegree at most one. We call each connected component of H' a* block *and all edges $e \in E_{intra}$* intra block edges*. A block B is* closed *if each vertex of B has indegree and outdegree one or* open*, otherwise. The* height *of B is defined as the number of intra block edges in B. The* cyclic block graph *$H = (V, E_{intra} \dot\cup E_{inter}, \phi)$ of H' is obtained by adding an edge $e \in E_{inter}$ from each vertex in H' to its consecutive right vertex on the same level (if there is one), which we call* inter block edges*.*

To create such a graph we first mark outer segments involved in type 1 conflicts between two levels. Then, we traverse the lower level from left to right and try to align each vertex with one of its median predecessor vertices. First we try its upper left median, then its upper right median. An alignment is impossible if the segment is marked or if it would cross a segment already used for aligning. The current vertex becomes the top vertex of a new block if both alignments fail. All inner segments of an edge are aligned and thus lie in the same block. As each block is drawn with constant slope, this ensures at most two bends per edge. E_{intra} is the set of all remaining edges. See Fig. 2 for an example. Vertices and intra block edges of the same block are framed. The inter block edges lie on the level lines. The dotted segments were removed in the block building phase.



(a) Left upper run (b) Right upper run

Fig. 2. Block graphs of Fig. 1(a) to (c) as 2D drawings

The cyclic block graph can have closed blocks (with height k) and open blocks with height $\geq k$ (spirals) which shall be avoided to simplify the algorithm (line 5). In both cases we split such a block by removing outer segments until each resulting block has at most height $k-1$. Such outer segments always exist, as no edge can span more than k levels. Therefore, the invariant of at most two bends per edge still holds. Note that an originally closed block will not be sheared like other blocks in Sect. 3.2 as it cannot be part of a cyclic dependency. See Fig. 2(b) for an example: The segment $(2, 4)$ was removed to open a closed block and the segment $(5, 7)$ was used to split a long block in two shorter ones. The result is a cyclic block graph with open blocks of height at most $k-1$.

3.2 Horizontal Compaction

In this section we compact the cyclic block graph by arranging all blocks as close to each other as possible minimizing the width of the drawing. Not all blocks can be drawn vertically as there can be cyclic dependencies in the left-to-right ordering among blocks, which we call rings.

Definition 2. *A* block path *P in a cyclic block graph $H = (V, E_{intra} \dot\cup E_{inter}, \phi)$ is a sequence of vertices $v_1, \ldots, v_s \in V$ s. t. for each pair of consecutive vertices v_i and $v_{i+1}, 1 \leq i < s$, $(v_i, v_{i+1}) \in E_{intra}$ or $(v_{i+1}, v_i) \in E_{intra}$ or $(v_i, v_{i+1}) \in E_{inter}$. It is* simple *if all vertices are mutually distinct. A block path is a* ring *R if $v_1 = v_s$. In a* simple ring *the vertices $v_1, \ldots, v_{s-1}$ are mutually distinct. The* width *of R is the number of inter block edges in R. Let c_{down} and c_{up} be the number of intra block edges traversed in R in and against their direction, respectively. The* number of windings *of R is then defined as* $\text{wind}(R) = (c_{down} - c_{up})/k$.

Informally, a ring is a cycle in the block graph where the direction of the inter block edges is preserved and the intra block edges are used in any direction.

wind(R) counts how often R wraps around the center. As each ring is an ordered sequence, we count windings along increasing and decreasing levels positive and negative, respectively. We consider the strongly connected components (SCCs) connected by block paths of the block graph separately. The *simple SCCs* consist of one block. All other *complex SCCs* contain rings. Figure 2(a) consists of three simple SCCs ($(2,4)$, $(7,9,12)$ and (13)) and one complex SCC (the remaining two blocks) in which all simple rings R have $\text{width}(R) = 2$ and $\text{wind}(R) = 1$.

Lemma 1. *For each ring R of a cyclic block graph G* $\text{wind}(R) \neq 0$.

Proof. Assume for contradiction that there exists a ring R with $\text{wind}(R) = 0$ in the block graph of G. Unwrapping G several times, i. e., placing multiple copies of the intermediate drawing one below the other and merging first and last levels, leads to a block graph of a (non-cyclic) level graph H which contains R completely. Then, H has a cyclic dependency, which is a contradiction. □

Lemma 2. *For each simple ring R of a cyclic block graph* $|\text{wind}(R)| \leq 1$.

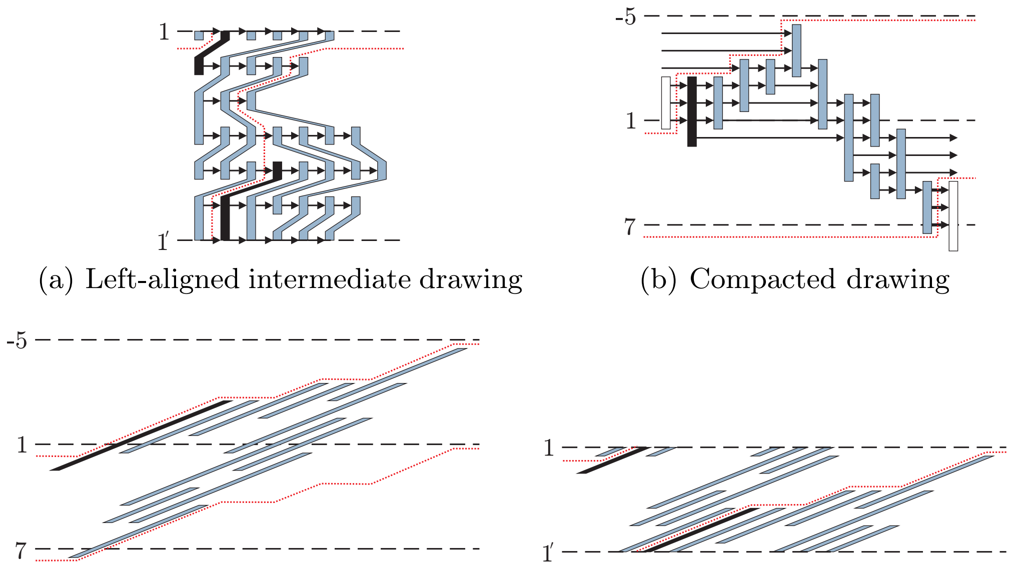
Proof. Assume for contradiction that there is a simple ring R with $|\text{wind}(R)| > 1$. As R wraps around the center in the 2D drawing more than once, each drawing of R crosses itself. Each cyclic path graph is planar. Further, each drawing of it respecting its ordering can be extended to a planar drawing of its cyclic block graph by adding the inter block edges along the level lines. Since R is a subgraph of the cyclic block graph, this is a contradiction. □

Theorem 1. *Let $\mathcal{R}$ be the set of all simple rings of an SCC in a cyclic block graph. For each $R \in \mathcal{R}$* $\text{wind}(R) = 1$ *or for each $R \in \mathcal{R}$* $\text{wind}(R) = -1$.

Proof. According to Lemmas 1 and 2 $|\text{wind}(R)| = 1$ holds for each ring $R \in \mathcal{R}$. Assume for contradiction that there exist two rings $R_1, R_2 \in \mathcal{R}$ with $\text{wind}(R_1) = 1$ and $\text{wind}(R_2) = -1$. Let v_1 and v_2 be vertices in R_1 and R_2, respectively. Let S be a (not necessarily simple) ring through v_1 and v_2, which always exists as v_1 and v_2 lie in the same SCC. Due to Lemma 1 $\text{wind}(S) \neq 0$. If $\text{wind}(S) > 0$, let T be a non-simple ring consisting of S and $\text{wind}(S)$ many copies of R_2 joined via v_2. Otherwise, let T be a ring consisting of S and $-\text{wind}(S)$ many copies of R_1 joined via v_1. In both cases $\text{wind}(T) = 0$, which contradicts Lemma 1. □

Definition 3. *Let S be a complex SCC of a cyclic block graph containing a simple ring R. We define* $\text{wind}(S) = \text{wind}(R)$ *and* $\text{width}(S)$ *as the maximum width of all simple rings in S.*

Horizontal Compaction of a complex Strongly Connected Component
It is not possible to draw all blocks of a complex SCC S straight-line and vertically. Therefore, we will draw all blocks of S with the same slope. The slope has to be chosen s. t. each ring, and thus the resulting curve, in S starts and ends at the same coordinates. All rings in S have the same number of windings wind(S), which is either 1 or -1. Thus, each simple ring spans $\text{wind}(S) \cdot k$ levels. To draw



(a) Left-aligned intermediate drawing (b) Compacted drawing

(c) Sheared drawing (d) Final intermediate drawing

Fig. 3. Drawing of a complex SCC

one simple ring R we could use the slope $-(\text{wind}(R) \cdot k)/\operatorname{width}(R)$, which would result in inter block edges of unit length. In order to draw all blocks in S with the same slope (line 10 in Algorithm 1) we must use the width of the widest simple ring of S and the slope $-(\text{wind}(S) \cdot k)/\operatorname{width}(S)$. With this slope the widest ring will fit exactly and use unit length inter block edges. All narrower rings will have some unused horizontal space in the drawing and thus have inter block edges which are longer than one unit.

To use this slope we have to determine the width of the widest simple ring in S. The general problem of finding a longest cycle is $\mathcal{NP}$-hard [4]. However, here it can be solved in linear time by cutting the SCC. Finding the length and compacting the layout is done simultaneously. To cut an SCC (line 8 in Algorithm 1), we start at an arbitrary block B and temporarily remove all incoming inter block edges of B. We then follow the outgoing inter block edge of the topmost vertex of B to the next block B'. We temporarily remove all incoming inter block edges of B' which are above the traversed incoming edge. We repeat this procedure until the topmost vertex of the current block has no outgoing inter block edge. Note that this happens before B is visited a second time, as otherwise the SCC would contain unreachable blocks. Thus, a block path P_r from B to a rightmost vertex in S is found. The same procedure is repeated from starting block B using the outgoing inter block edge of the lowest vertex to reach B' and deleting all incoming inter block edges below the traversed edge until a lowest vertex with no incoming inter block edge is found, which gives the block path P_l.

Combining P_r and P_l results in a y-monotone path P from a rightmost vertex through B to a leftmost vertex, using inter block edges in both directions and preserving intra block edge directions. Due to the removed incoming inter block edges left of P all rings in S are cut exactly once. We then assign an arbitrary vertex $v \in V$ of B the new coordinate $y'(v) = \phi(v)$. In a traversal of the block graph we assign each vertex a y'-coordinate: Using an inter block edge we assign both end vertices the same y'-coordinate. Using an intra block edge in or against its direction we increase or decrease the y'-coordinate by 1, respectively, without using a modulo operation. The result is an acyclic block graph, which we compact (in contrast to [3]) in the following way (line 9 in Algorithm 1): We place each block which is a source in the acyclic block graph on an imaginary zero line, treat all other blocks in the topological order and move them as much to the left as possible preserving unit distance. Afterwards, we fix all sinks on their positions, treat all other blocks against the topological order and move them as much to the right as possible. For placing a block as close as possible to the already placed ones, we traverse its levels. After the compaction each block (and therefore each vertex v in S) has an assigned x'-coordinate. Let $e = (u, v)$ be a removed inter block edge. The width of the widest simple ring of S through e is then $x'(u) - x'(v) + 1$. Considering all removed inter block edges and computing the maximum value gives the width of the widest simple ring width(S) in S.

See Fig. 3 for an example of an SCC S with wind$(S) = 1$ and $k = 6$ levels. Figure 3(a) shows a leftmost intermediate drawing with the black start block B. Its lowest vertex is already the leftmost vertex on its level. To reach a rightmost vertex four other blocks have to be visited. The dashed line cuts six inter block edges. Figure 3(b) shows the resulting compacted horizontal block graph using y'-coordinates. The widths of the widest rings through each of the six cutted edges are (from top to bottom): 5, 5, 7, 10, 10, 10. Thus, width$(S) = 10$. Shearing the drawing with slope $\frac{-1 \cdot 6}{10}$ results in Fig. 3(c). Using the modulo operation for the y-coordinates gives the final intermediate drawing in Fig. 3(d).

Theorem 2. *The intermediate drawing of S uses the coordinates $x(v) = x'(v) - (\text{width}(S)/(\text{wind}(S) \cdot k)) \cdot y'(v)$ and $y(v) = ((y'(v) - 1) \bmod k) + 1 = \phi(v)$ for each vertex v in S. In the drawing all intra block edges have* slope$(S) = -(\text{wind}(S) \cdot k)/\text{width}(S)$*. The ordering of vertices on the same level is the one given by the crossing reduction phase and these vertices have at least unit distance.*

Proof. In the compacted drawing of the open block graph all blocks are drawn vertically. Shearing the drawing results in unchanged y'-coordinates and new x-coordinates $x(v) = x'(v) + y'(v)/\text{slope}(S)$. Now all edges have slope slope(S). Using the y-coordinates $y(v) = ((y'(v) - 1) \mod k) + 1 = \phi(v)$ does not affect the slope of the edges. But now all vertices on the same level have the same y-coordinate again. Let u and v be two consecutive vertices on the same level. Let u be left of v according to the crossing reduction. If the inter block edge (u, v) was not cut before, then u and v have the same y'-coordinates in the compacted drawing and u is still the left neighbor of v with at least unit distance between them. This does not change in the sheared or final intermediate drawing. If (u, v) was cut, then $y'(v) = y'(u) - k \cdot \text{wind}(S)$. There exists a simple block path P from v to u as we are compacting an SCC. P cannot have been cut as otherwise P and (u, v) form a simple ring that would have been cut twice. The ring formed by P and (u, v) is at most width(S) wide and thus $x'(v) \geq x'(u) - (\text{width}(S) - 1)$.

After the drawing is sheared, $x(v) \geq x(u) + 1$ holds. Therefore, u is still left of v and the two vertices are at least unit distance apart. As a result, all consecutive vertices and thus all vertices on the same level are separated by unit distance and are in the ordering given by the crossing reduction phase. □

Horizontal Compaction Our next step is to globally compact the set of compacted complex SCCs and simple SCCs (line 12 in Algorithm 1).

Lemma 3. *In a drawing that respects the order of the crossing reduction phase all vertices of an SCC on the same level are consecutive.*

Proof. Let u, v be two vertices of an SCC on the same level s. t. u is left of v. Note that there is a block path from v to u. Also, there is a horizontal path from u to v using inter block edges only. Therefore, all vertices between u and v lie on a ring containing u and v and thus belong to the same SCC. □

This means that no SCCs can interleave. We interpret the SCCs as super vertices and perform a topological sorting on the resulting DAG. We then compact the SCCs as we compacted the blocks of a (non-cyclic) block graph before.

3.3 Balancing

In this phase (line 14 in Algorithm 1) the four results are balanced by computing one x-coordinate for each vertex out of the four x-coordinates computed by the four runs. The only difference to the algorithm of Brandes and Köpf [3] is that we do not use the average median of the four x-coordinates for each vertex, since this can induce additional bends in the cyclic case. The reason is that on lines with different slopes the median changes at crossings, i. e., it is a non-linear function. Hence, we use the average of all four x-coordinates for each vertex.

Proposition 1. *Using the average of the x-coordinates of the four runs for each vertex does not change the ordering of the vertices on a level and preserves at least unit distance. Additional bends can occur since the blocks of the four runs may differ. However, the invariant of at most two bends per edge e in the final drawing still holds, as the y-coordinates of the bends located at the topmost and lowest dummy vertex of e are identical in each drawing.*

Note that it is possible that some vertices in one run belong to a block of an SCC although they do not belong to an SCC or even one block in another run. Thus, balancing can lead to more different slopes than in each of the four runs alone. See Fig. 2 for two different block graphs of two runs of the same graph.

4 Algorithm Analysis

Theorem 3. *Let $G = (V, E, \phi, \prec)$ be a proper ordered cyclic k-level graph. The width of the intermediate drawing of G is $\mathcal{O}(|V|^2/k)$ and the area is $\mathcal{O}(|V|^2)$. For the 3D drawing the same bounds hold. The 2D drawing has a width and height of $\mathcal{O}(|V|^2/k)$ and thus an area of $\mathcal{O}(|V|^4/k^2)$.*

Proof. Let $\mathcal{S} = \{S_1, \ldots, S_r\}$ be the set of all SCCs. Let N_i be the number of (dummy) vertices in S_i. The width of the compacted drawing of S_i is $\text{width}(S_i) \leq N_i$. Shearing the drawing of height at most N_i with slope $-\text{wind}(S) \cdot k/\text{width}(S_i)$ adds at most N_i^2/k to the width. Thus, the width of the drawing is in $\mathcal{O}(N_i^2/k)$. The width of the drawing of G is at most the sum of the widths of the drawings of all SCCs and thus in $\mathcal{O}(|V|^2/k)$. As the height is k, the area is in $\mathcal{O}(|V|^2)$.

The height and width of the 2D drawing is twice the width of the intermediate drawing and thus the area is $\mathcal{O}(|V|^4/k^2)$, however. □

Note that the width of the drawing reduces to $\mathcal{O}(|V|)$ if there are no complex SCCs in the graph. This reduces the area of the intermediate and 3D drawings to $\mathcal{O}(|V| \cdot k)$ and of the 2D drawing to $\mathcal{O}(|V|^2)$. Complex SCCs can always be avoided by using special crossing reduction and block building algorithms [5].

Theorem 4. *The layout algorithm described in Algorithm 1 has a time complexity of $\mathcal{O}(|V| + |E|)$ for a proper ordered cyclic k-level graph $G = (V, E, \phi, <)$.*

5 Summary

We presented the first coordinate assignment algorithm for cyclic level graphs. Like the established algorithm by Brandes and Köpf, which is the de facto standard coordinate assignment method for hierarchic level graphs, we ensure to have at most two bends per edge and try to align long edges and center parents over their children. These are the major aesthetic criteria for such drawings. We implemented a prototype of our algorithm within the Gravisto toolkit [2].

References

1. Bachmaier, C., Brandenburg, F.J., Brunner, W., Lovász, G.: Cyclic leveling of directed graphs. In: Tollis, I., Patrignani, M. (eds.) GD 2008. LNCS, vol. 5417, pp. 348–359. Springer (2009)
2. Bachmaier, C., Brandenburg, F.J., Forster, M., Holleis, P., Raitner, M.: Gravisto: Graph visualization toolkit. In: Pach, J. (ed.) GD 2004. LNCS, vol. 3383, pp. 502–503. Springer (2004), `http://gravisto.fim.uni-passau.de/`
3. Brandes, U., Köpf, B.: Fast and simple horizontal coordinate assignment. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) GD 2001. LNCS, vol. 2265, pp. 31–44. Springer (2001)
4. Garey, M.R., Johnson, D.S.: A Guide to the Theory of NP-Completeness. W. H. Freemann, New York (1979)
5. Hübner, F.: A Global Approach on Crossing Minimization in Hierarchical and Cyclic Layouts of Leveled Graphs. Master's thesis, University of Passau (2009)
6. Kaufmann, M., Wagner, D.: Drawing Graphs, LNCS, vol. 2025. Springer (2001)
7. Michal, G. (ed.): Biochemical Pathways: An Atlas of Biochemistry and Molecular Biology. Wiley (1998)
8. Serafini, P., Ukovich, W.: A mathematical model for periodic scheduling problems. SIAM J. Discrete Math. 2(4), 550–581 (1989)
9. Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Trans. Syst., Man, Cybern. 11(2), 109–125 (1981)

List of Figures

Bibliography

[1] C. Bachmaier. *Circle Planarity of Level Graphs*. Dissertation, Faculty of Informatics and Mathematics, University of Passau, 2004.

[2] C. Bachmaier. A radial adaption of the sugiyama framework for visualizing hierarchical information. *IEEE Transactions on Visualization and Computer Graphics*, 13(3):583–594, 2007.

[3] C. Bachmaier, F. J. Brandenburg, W. Brunner, and R. Fülöp. Coordinate assignment for cyclic level graphs. In H. Q. Ngo, editor, *Proc. Computing and Combinatorics, COCOON 2009*, volume 5609 of *LNCS*, pages 66–75. Springer, 2009.

[4] C. Bachmaier, F. J. Brandenburg, W. Brunner, and F. Hübner. A global k-level crossing reduction algorithm. In M. S. Rahman and S. Fujita, editors, *Proc. Workshop on Algorithms and Computation, WALCOM 2010*, volume 5942 of *LNCS*, pages 70–81. Springer, 2010.

[5] C. Bachmaier, F. J. Brandenburg, W. Brunner, and G. Lovász. Cyclic leveling of directed graphs. In I. G. Tollis and M. Patrignani, editors, *Proc. Graph Drawing, GD 2008*, volume 5417 of *LNCS*, pages 348–349. Springer, 2009.

[6] C. Bachmaier, F. J. Brandenburg, and M. Forster. Radial level planarity testing and embedding in linear time (extended abstract). In G. Liotta, editor, *Proc. Graph Drawing, GD 2003*, volume 2912 of *LNCS*, pages 393–405. Springer, 2004.

[7] C. Bachmaier, F. J. Brandenburg, and M. Forster. Radial level planarity testing and embedding in linear time. *Journal of Graph Algorithms and Applications*, 9(5):53–97, 2005.

[8] C. Bachmaier, F. J. Brandenburg, M. Forster, P. Holleis, and M. Raitner. Gravisto: Graph visualization toolkit. In J. Pach, editor, *Proc. Graph Drawing, GD 2004*, volume 3383 of *LNCS*, pages 502–503. Springer, 2004.

[9] C. Bachmaier and W. Brunner. Linear time planarity testing and embedding of strongly connected cyclic level graphs. In D. Halperin and K. Mehlhorn, editors, *Proc. European Symposium on Algorithms, ESA 2008*, volume 5193 of *LNCS*, pages 136–147. Springer, 2008.

[10] C. Bachmaier, W. Brunner, and C. König. Cyclic level planarity testing and embedding (extended abstract). In S.-H. Hong, T. Nishizeki, and W. Quan, editors, *Proc. Graph Drawing, GD 2007*, volume 4875 of *LNCS*, pages 50–61. Springer, 2007.

[11] C. Bachmaier, H. Buchner, M. Forster, and H. Seok-Hee. Crossing minimization in extended level drawings of graphs. *Discrete Applied Mathematics*, 158(3):159–179, 2010.

[12] C. Bachmaier, F. Fischer, and M. Forster. Radial coordinate assignment for level graphs. In L. Wang, editor, *Proc. Computing and Combinatorics, COCOON 2005*, volume 3595 of *LNCS*, pages 401–410. Springer, 2005.

[13] C. Bachmaier and M. Forster. Crossing reduction for hierarchical graphs with intra-level edges. Technical Report MIP-0608, University of Passau, 2006.

[14] C. Bachmaier and M. Forster. A radial adaption of the Sugiyama framework for hierarchical graph drawing. Technical Report MIP-0603, University of Passau, April 2006.

[15] W. Barth, M. Jünger, and P. Mutzel. Simple and efficient bilayer cross counting. In M. T. Goodrich and S. G. Kobourov, editors, *Proc. Graph Drawing, GD 2002*, volume 2528 of *LNCS*, pages 130–141. Springer, 2002.

[16] M. Baur and U. Brandes. Crossing reduction in circular layout. In J. Hromkovic, M. Nagl, and B. Westfechtel, editors, *Proc. Workshop on Graph-Theoretic Concepts in Computer Science, WG 2004*, volume 3353 of *LNCS*, pages 332–343. Springer, 2004.

[17] M. Baur and U. Brandes. Multi-circular layout of micro/macro graphs. In S. Hong, T. Nishizeki, and W. Quan, editors, *Proc. Graph Drawing, GD 2007*, volume 4875 of *LNCS*, pages 255–267. Springer, 2008.

[18] B. Berger and S. Peter. Approximation algorithms for the maximum acyclic subgraph problem. In *Proc. ACM-SIAM Symposium on Discrete Algorithms, SODA 1990*, pages 236–243. SIAM, 1990.

[19] M. Berkelaar, J. Dirks, J. Ebert, K. Eikland, and P. Notebaert. lp_solve: Mixed integer linear programming solver. http://lpsolve.sourceforge.net/.

[20] S. Borgatti, S. Kobourov, O. Kohlbacher, and P. Mutzel. Graph drawing with applications to bioinformatics and social sciences, Schloss Dagstuhl seminar 08191, Mai 2008.

[21] U. Brandes and T. Erlebach, editors. *Network Analysis: Methodological Foundations*, volume 3418 of *LNCS Tutorial*. Springer, 2005.

[22] U. Brandes, P. Kenis, and D. Wagner. Centrality in policy network drawings. In J. Kratochvíl, editor, *Proc. Graph Drawing, GD 1999*, volume 1731 of *LNCS*, pages 250–258. Springer, 1999.

[23] U. Brandes, P. Kenis, and D. Wagner. Communicating centrality in policy network drawings. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):241–253, 2003.

[24] U. Brandes and B. Köpf. Fast and simple horizontal coordinate assignment. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Proc. Graph Drawing, GD 2001*, volume 2265 of *LNCS*, pages 31–44. Springer, 2002.

[25] U. Brandes and C. Pich. More flexible radial layouts. In D. Eppstein and E. R. Gansner, editors, *Proc. Graph Drawing, GD 2009*, volume 5849 of *LNCS*, pages 107–118. Springer, 2010.

[26] U. Brandes, J. Raab, and D. Wagner. Exploratory network visualization: Simultaneous display of actor status and connections. *Journal of Social Structure*, 2(4), 2001.

[27] J. Branke, S. Leppert, and P. Eades. Width-restricted layering of acyclic digraphs with consideration of dummy nodes. *Information Processing Letters*, 81:59–63, 2002.

[28] C. Buchheim, M. Jünger, and S. Leipert. A fast layout algorithm for k-level graphs. In J. Marks, editor, *Proc. Graph Drawing, GD 2000*, volume 1984 of *LNCS*, pages 229–240. Springer, 2001.

[29] C. Buchheim and L. Zheng. A new exact algorithm for the two-sided crossing minimization problem. In A. Dress, Y. Xu, and B. Zhu, editors, *Proc. Conference on Combinatorial Optimization and Application, COCOA 2007*, volume 4616 of *LNCS*, pages 301–310. Springer, 2007.

[30] H. Buchner. Displaying centralities using orbital layout. Diploma thesis, Faculty of Informatics and Mathematics, University of Passau, 2006.

[31] M.-J. Carpano. Automatic display of hierarchized graphs for computer aided decision analysis. *IEEE Transactions on Systems, Man, and Cybernetics*, 10(11):705–715, 1980.

[32] M. Chimani, C. Gutwenger, M. Jünger, K. Klein, P. Mutzel, and M. Schulz. Open Graph Drawing Framework, OGDF. http://www.ogdf.net/.

[33] M. Chimani, C. Gutwenger, P. Mutzel, and H.-M. Wong. Layer-free upward crossing minimization. In C. C. McGeoch, editor, *Proc. Workshop on Experimental Algorithms, WEA 2008*, volume 5038 of *LNCS*, pages 55–68. Springer, 2008.

[34] M. Chimani, C. Gutwenger, P. Mutzel, and H.-M. Wong. Upward planarization layout. In D. Eppstein and E. R. Gansner, editors, *Proc. Graph Drawing, GD 2009*, volume 5849 of *LNCS*, pages 94–106. Springer, 2010.

[35] J. Clark and D. A. Holton. *Graphentheorie, Grundlagen und Anwendungen*. Spektrum, 1994.

[36] E. G. Coffman Jr. and R. L. Graham. Optimal scheduling for two processor systems. *Acta Informatica*, 1(3):200–213, 1972.

[37] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

[38] M. C. F. de Oliveira and H. Levkowitz. From visual data exploration to visual data mining: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):378–394, 2003.

[39] C. Demetrescu and I. Finocchi. Break the „right" cycles and get the „best" drawing. In A. V. Goldberg and B. M. E. Moret, editors, *Proc. Workshop on Algorithms Engineering and Experiments, ALENEX 2000*, pages 171–182, 2000.

[40] C. Demetrescu and I. Finocchi. Breaking cycles for minimizing crossings. *ACM Journal on Experimental Algorithms*, 6(1):1–39, 2001.

[41] C. Demetrescu and I. Finocchi. Combinatorial algorithms for feedback problems in directed graphs. *Information Processing Letters*, 86(3):129–136, 2003.

[42] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.

[43] E. Di Giacomo, W. Didimo, L. Giuseppe, and H. Meijer. Computing radial drawings on the minimum number of circles. In J. Pach, editor, *Proc. Graph Drawing, GD 2004*, volume 3383 of *LNCS*, pages 2651–261. Springer, 2005.

[44] E. Di Giacomo, W. Didimo, L. Giuseppe, and H. Meijer. Computing radial drawings on the minimum number of circles. *Journal of Graph Algorithms and Applications*, 9(3):347–364, 2005.

[45] E. Di Giacomo, W. Didimo, and G. Liotta. Radial drawings of graphs: Geometric constraints and trade-offs. In M. Kaufmann and D. Wagner, editors, *Proc. Graph Drawing, GD 2006*, volume 4372 of *LNCS*, pages 355–366. Springer, 2007.

[46] E. Di Giacomo, W. Didimo, and G. Liotta. Radial drawings of graphs: Geometric constraints and trade-offs. *Journal of Discrete Algorithms*, 6(1):109–124, 2008.

[47] R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, second edition, 2000. electronic version from `ftp://ftp.math.uni-hamburg.de/pub/unihh/math/books/diestel/GraphTheoryII.pdf`.

[48] P. Doreian and L. H. Albert. Partitioning political actor networks: Some quantitative tools for analyzing qualitative networks. *Journal of Quantitative Anthropology*, 1:279–291, 1989.

[49] P. Eades. Drawing free trees. *Bulletin of the Institute of Combinatorics and its Applications*, 5:10–36, 1992.

[50] P. Eades, Q.-W. Feng, and X. Lin. Straight-line drawing algorithms for hierarchical graphs and clustered graphs. In S. C. North, editor, *Proc. Graph Drawing, GD 1996*, volume 1190 of *LNCS*, pages 113–128. Springer, 1997.

[51] P. Eades, Q.-W. Feng, X. Lin, and H. Nagamochi. Straight-line drawing algorithms for hierarchical graphs and clustered graphs. *Algorithmica*, 44(1):1–32, 2006.

[52] P. Eades and D. Kelly. Heuristics for reducing crossings in 2-layered networks. *Ars Combinatorica*, 21(A):89–98, 1986.

[53] P. Eades, X. Lin, and W. F. Smyth. A fast and effective heuristic for the feedback arcset problem. *Information Processing Letters*, 47:319–323, 1993.

[54] P. Eades, X. Lin, and R. Tamassia. An algorithm for drawing a hierarchical graph. *International Journal of Computational Geometry and Applications*, 6:145–156, 1996.

[55] P. Eades, B. D. McKay, and N. C. Wormald. On an edge crossing problem. In *Proc. Australian Computer Science Conference*, pages 327–334. Australian National University, 1986.

[56] P. Eades and K. Sugiyama. How to draw a directed graph. *Journal of Information Processing*, 13(4):424–437, 1990.

[57] P. Eades and S. H. Whitesides. Drawing graphs in two layers. *Theoretical Computer Science*, 131:361–374, 1994.

[58] P. Eades and N. C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11(1):379–403, 1994.

[59] M. Eiglsperger, M. Siebenhaller, and M. Kaufmann. An efficient implementation of Sugiyama's algorithm for layered graph drawing. *Journal of Graph Algorithms and Applications*, 9(3):305–325, 2005.

[60] M. Eiglsperger, M. Siebenhaller, and M. Kaufmann. An efiicient implementation of Sugiyama's algorithm for layered graph drawing. In J. Pach, editor, *Proc. Graph Drawing, GD 2004*, volume 3383 of *LNCS*, pages 155–166. Springer, 2005.

[61] J. Ellson, E. R. Gansner, L. Koutsofios, S. C. North, and G. Woodhull. Graphviz – open source graph drawing tools. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Proc. Graph Drawing, GD 2001*, volume 2265 of *LNCS*, pages 594–597. Springer, 2002.

[62] T. Eschbach, W. Günther, R. Drexler, and B. Becker. Crossing reduction by windows optimization. In M. T. Goodrich and S. G. Kobourov, editors, *Proc. Graph Drawing, GD 2002*, volume 2528 of *LNCS*, pages 285–294. Springer, 2002.

[63] G. Even, J. S. Naor, B. Shieber, and M. Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998.

[64] J.-P. Ewert and S. B. Ewert. *Wahrnehmung*. Biologische Arbeitsbücher 35. Quelle & Meyer, 1981.

[65] U. Fayyad, G. G. Grinstein, and A. Wierse, editors. *Information Visualization in Data Mining and Knowledge Discovery*, chapter 2 A Survey of Visualizations for High-Dimensional Data Mining. Morgan Kaufmann, 2002.

[66] F. Fischer. Koordinatenzuweisung beim radialen Zeichnen von gerichteten Graphen. Diplomarbeit, Fakultät für Informatik und Mathematik, Universität Passau, 2004.

[67] R. Fülöp. Erweiterung des Brandes/Köpf-Algorithmus auf zyklische Graphen. Diplomarbeit, Fakultät für Informatik und Mathematik, Universität Passau, 2008.

[68] M. Forster and C. Bachmaier. Clustered level planarity. In P. Van Emde Boas, J. Pokorný, M. Bieliková, and J. Štuller, editors, *Proc. Software Seminar: Theory and Practice of Informatics, SOFSEM 2004*, volume 2932 of *LNCS*, pages 218–228. Springer, 2004.

[69] A. Frick. Upper bounds on the number of hidden nodes in sugiyama's algorithm. In S. C. North, editor, *Proc. Graph Drawing, GD 1996*, volume 1190 of *LNCS*, pages 169–183. Springer, 1997.

[70] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software – Practice and Experience*, 21(11):1129–1164, 1991.

[71] M. K. Ganapathy and S. P. Lodha. On minimum circular arrangement. In V. Diekert and M. Habib, editors, *Proc. Symposium on Theoretical Aspects of Computer Science, STACS 2004*, volume 2996 of *LNCS*, pages 394–405. Springer, 2004.

[72] E. R. Gansner, Y. Koren, and S. North. Graph drawing by stress majorization. In J. Pach, editor, *Proc. Graph Drawing, GD 2004*, volume 3383 of *LNCS*. Springer, 2004.

[73] E. R. Gansner, E. Koutsofios, S. C. North, and K.-P. Vo. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering*, 19(3):214–230, 1993.

[74] E. R. Gansner, S. C. North, and K.-P. Vo. DAG, a program that draws directed graphs. *Software – Practice and Experience*, 17(1):1047–1062, 1988.

[75] M. R. Garey and D. S. Johnson. *A Guide to the Theory of $\mathcal{NP}$-Completeness*. Freemann, 1979.

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

[77] W. Günther, R. Schönfeld, B. Becker, and P. Molitor. k-layer straightline crossing minimization by speeding up sifting. In J. Marks, editor, *Proc. Graph Drawing, GD 2000*, volume 1984 of *LNCS*, pages 253–258. Springer, 2001.

[78] M. Grötschel, M. Jünger, and G. Reinelt. On the acyclic subgraph polytope. *Mathematical Programming*, 33(1):28–42, 1985.

[79] C. Gutwenger, M. Jünger, J. Klein, Karsten Kupke, S. Leipert, and P. Mutzel. GoVisual: A diagramming software for uml class diagrams. In M. Jünger and P. Mutzel, editors, *Graph Drawing Software*, Mathematics and Visualization, pages 257–278. Springer, 2004.

[80] M. Harrigan and P. Healy. Practical level planarity testing and layout with embedding constraints. In S.-H. Hong, T. Nishizeki, and W. Quan, editors, *Proc. Graph Drawing, GD 2007*, volume 4875 of *LNCS*, pages 62–68. Springer, 2008.

[81] F. Hübner. A global approach on crossing minimization in hierarchical and cyclic layouts of leveled graphs. Diploma thesis, Faculty of Informatics and Mathematics, University of Passau, 2009.

[82] P. Healy and A. Kuusik. The vertex-exchange graph: A new concept for multi-level crossing minimisation. In J. Kratochvíl, editor, *Proc. Graph Drawing, GD 1999*, volume 1731 of *LNCS*, pages 205–216. Springer, 1999.

[83] P. Healy and A. Kuusik. Algorithms for multi-level graph planarity testing and layout. *Theoretical Computer Science*, 320(2–3):331–344, 2004.

[84] P. Healy and N. S. Nikolov. How to layer a directed acyclic graph. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Proc. Graph Drawing, GD 2001*, volume 2265 of *LNCS*, pages 16–30. Springer, 2002.

[85] M. Höllmüller. Radiales Zeichnen von sozialen Netzwerken. Diplomarbeit, Fakultät für Informatik und Mathematik, Universität Passau, 2006.

[86] S.-H. Hong and H. Nagamochi. Approximating crossing minimization in radial layouts. In E. Sany Laber, C. F. Bornstein, L. T. Nogueira, and L. Faria, editors, *Proc. Latin American Theoretical Informatics Symposium, LATIN 2008*, volume 4957 of *LNCS*, pages 461–472. Springer, 2008.

[87] S.-H. Hong and H. Nagamochi. New approximation to the one-sided radial crossing minimization. *Journal of Graph Algorithms and Applications*, 13(2):179–196, 2009.

[88] S.-H. Hong and H. Nagamochi. Approximation algorithms for minimizing edge crossings in radial drawings. *Algorithmica*, 58:478–497, 2010.

[89] T. J. Jankun-Kelly and K.-L. Ma. Moiregraphs: Radial focus+context visualization and interaction for graphs with visual nodes. In T. Munzner and S. North, editors, *Proc. IEEE Symposium on Information Visualization, INFOVIS 2003*, pages 59–66. IEEE Computer Society Press, 2003.

[90] M. Jünger, E. K. Lee, P. Mutzel, and T. Odenthal. A polyhedral approach to the multi-layer crossing minimization problem. In G. Di Battista, editor, *GD 1997*, volume 1353 of *LNCS*, pages 13–24. Springer, 1997.

[91] M. Jünger and S. Leipert. Level planar embedding in linear time. *Journal of Graph Algorithms and Applications*, 6(1):67–113, 2002.

[92] M. Jünger and P. Mutzel. 2-layer straight line crossing minimization: Performance of exact and heuristic algorithms. *Journal of Graph Algorithms and Applications*, 1:1–25, 1997.

[93] D. S. Johnson. The $\mathcal{NP}$-completeness column: An ongoing guide. *Journal of Algorithms*, 3(1):89–99, 1982.

[94] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989.

[95] R. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[96] M. Kaufmann and D. Wagner. *Drawing Graphs*, volume 2025 of *LNCS*. Springer, 2001.

[97] Y. Koren. On spectral graph drawing. In T. Warnow and B. Zhu, editors, *Proc. Computing and Combinatorics, COCOON 2003*, volume 2697 of *LNCS*, pages 496–508. Springer, 2003.

[98] A. Kramer. Auf einen Blick: Komplexe Informationen in Sekunden vermitteln. *c't magazin für computer und technik*, 4:154–159, 2009.

[99] P. Kuntz, B. Pinaud, and R. Lehn. Minimizing crossings in hierarchical digraphs with a hybridized genetic algorithm. *Journal of Heuristics*, 12(1–2):23–26, 2006.

[100] M. Laguna, R. Martí, and V. Valls. Arc crossing minimization in hierarchical digraphs with tabu search. *Computers & Operations Research*, 24(12):1175–1186, 1997.

[101] S. Lam and R. Sethi. Worst case analysis of two scheduling problems. *SIAM Journal on Computing*, 6(3):518–536, 1977.

[102] J. Y.-T. Leung, editor. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Computer and Information Science. CRC, 2004.

[103] V. Liberatore. Circular arrangements. In P. Widmayer, F. Triguero, R. Morales, M. Hennessy, S. Eidenbenz, and R. Conejo, editors, *Proc. International Colloquium on Automata, Languages, and Programming, ICALP 2002*, volume 2380 of *LNCS*, pages 1054–1065. Springer, 2002.

[104] V. Liberatore. Multicast scheduling for list requests. In *Proc. Joint Conference of the IEEE Computer and Communications Societies, Infocom 2002*, volume 2, pages 1129–1137. IEEE, 2002.

[105] V. Liberatore. Circular arrangements and cyclic broadcast scheduling. *Journal of Algorithms*, 51(2):185–215, 2004.

[106] G. Lovász. Entwurf und Implementierung von Algorithmen zur Berechnung einer Ebeneneinteilung bei zyklischen Levels. Diplomarbeit, Fakultät für Informatik und Mathematik, Universität Passau, 2008.

[107] S. Masuda, T. Kashiwabara, K. Nakajima, and T. Fujisawa. On the $\mathcal{NP}$-completeness of a computer network layout problem. In *Proc. IEEE International Symposium on Circuits and Systems*, pages 292–295, 1987.

[108] C. Matuszewski, R. Schönfeld, and P. Molitor. Using sifting for k-layer straightline crossing minimization. In J. Kratochvíl, editor, *Proc. Graph Drawing, GD 1999*, volume 1731 of *LNCS*, pages 217–224. Springer, 1999.

[109] K. Mehlhorn and W. Rülling. Compaction on the torus. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 9:389–397, 1990.

[110] G. Michal, editor. *Biochemical Pathways: An Atlas of Biochemistry and Molecular Biology*. Wiley, 1998.

[111] E. Mäkinen. On circular layouts. *International Journal of Computer Mathematics*, 24:29–37, 1988.

[112] X. Muñoz, W. Unger, and I. Vrt'o. One sided crossing minimization is $\mathcal{NP}$-hard for sparse graphs. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Proc. Graph Drawing, GD 2001*, volume 2265 of *LNCS*, pages 115–122. Springer, 2002.

[113] P. Mutzel. An alternative method to crossing minimization on hierarchical graphs. In S. C. North, editor, *Proc. Graph Drawing, GD 1996*, volume 1190 of *LNCS*, pages 318–333. Springer, 1997.

[114] P. Mutzel. An alternative method to crossing minimization on hierarchical graphs. *SIAM Journal on Optimization*, 11(4):1065–1080, 2001.

[115] P. Mutzel and R. Weiskircher. Two-layer planarization in graph drawing. In K.-Y. Chwa and O. H. Ibarra, editors, *Proc. International Symposium on Algorithms and Computation, ISAAC 1998*, volume 1533 of *LNCS*, pages 69–79. Springer, 1998.

[116] H. Nagamochi. An improved bound on the one-sided minimum crossing number in two-layered drawing. *Discrete & Computational Geometry*, 33:569–591, 2005.

[117] J. S. Naor and R. Schwartz. The directed circular arrangement problem. In *Proc. ACM-SIAM Symposium on Discrete Algorithms, SODA 2004*, pages 85–95. SIAM, 2004.

[118] S. C. North, editor. *The Graphlet System (System Demonstration)*, volume 1190 of *LNCS*. Springer, 1997.

[119] T. Ottmann and P. Widmayer. *Algorithmen und Datenstrukturen*. Spektrum, third edition, 1996.

[120] C. Pich. Clockwise drawing of directed graphs. Unpublished Preprint, Department of Computer & Information Science, University of Konstanz, April 2009.

[121] C. Pich. Drawing directed graphs clockwise. In D. Eppstein and E. R. Gansner, editors, *Proc. Graph Drawing, GD 2009*, volume 5849 of *LNCS*, pages 369–380. Springer, 2010.

[122] H. C. Purchase. Which aesthetic has the greatest effect on human understanding? In G. Di Battista, editor, *Proc. Graph Drawing, GD 1997*, volume 1353 of *LNCS*, pages 248–261. Springer, 1997.

[123] H. C. Purchase. Metrics for graph drawing aesthetics. *Journal of Visual Languages and Computing*, 13:501–516, 2002.

[124] B. Randerath, E. Speckenmeyer, E. Boros, P. Hammer, A. Kogan, K. Makino, B. Simeone, and O. Cepek. A satisfiability formulation of problems on level graphs. *Electronic Notes in Discrete mathematics*, 9:269–277, 2001.

[125] M. G. Reggiani and F. E. Marchetti. A proposed method for representing hierarchies. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(1):2–8, 1988.

[126] L. A. Rowe, M. Davis, E. Mesinger, C. Meyer, C. Spirkais, and A. Tuan. A browser for directed graphs. *Software – Practice and Experience*, 17:61–76, 1998.

[127] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proc. IEEE/ACM International Conference on Computer Aided Design, ICCAD 1993*, pages 42–47. IEEE Comp. Society Press, 1993.

[128] Y. Saab. A fast and effective algorithm for the feedback arcset problem. *Journal of Heuristics*, 7:235–250, 2001.

[129] G. Sander. Graph layout through the VCG tool. In R. Tamassia and I. G. Tollis, editors, *Proc. Graph Drawing, GD 1994*, volume 894 of *LNCS*, pages 194–205. Springer, 1995.

[130] G. Sander. A fast heuristic for hierarchical manhattan layout. In F. J. Brandenburg, editor, *Proc. Graph Drawing, GD 1995*, volume 1027 of *LNCS*, pages 447–458. Springer, 1996.

[131] G. Sander. Graph layout for applications in compiler construction. *Theoretical Computer Science*, 217:175–214, 1999.

[132] P. Serafini and W. Ukovich. A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematics*, 2(4):550–581, 1989.

[133] F. Shahrokhi, O. Sýkora, L. A. Székly, and I. Vrt'o. On bipartite drawings and the linear arrangement problem. *SIAM Journal on Computing*, 30(1973–1789), 2001.

[134] J. M. Six and I. G. Tollis. A framework and algorithms for circular drawings of graphs. *Journal of Discrete Algorithms*, 4(1):25–50, 2006.

[135] D. D. Sleater and R. E. Tarjan. Self-adjusting binary search trees. *Journal of the Association for Computing Machinery*, 3:652–686, 1985.

[136] K. Sugiyama. *Graph Drawing and Applications for Software and Knowledge Engineers*, volume 11 of *Software Engineering and Knowledge*. World Scientific, 2002.

[137] K. Sugiyama and K. Misue. A simple and unified method for drawing graphs: Magnetic-spring algorithm. In R. Tamassia and I. G. Tollis, editors, *Proc. Graph Drawing, GD 1994*, volume 894 of *LNCS*, pages 364–375. Springer, 1995.

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

[139] N. Tomii, Y. Kambayashi, and S. Yajima. On planarization of 2-level graphs. *Papers of Technical Group on Electronic Computers, IECEJ*, EC77-38:1–12, 1977.

[140] J. Utech, J. Branke, H. Schmeck, and P. Eades. An evolutionary algorithm for drawing directed graphs. In H. R. Arabnia, editor, *Proc. International Conference on Imaging Science, Systems, and Technology, CISST 1998*, pages 154–160. CSREA, 1998.

[141] V. Valls, R. Martí, and P. Lino. A branch and bound algorithm for minimizing the number of crossing arcs in bipartite graphs. *Journal of Operational Research*, 90:303–319, 1996.

[142] V. Waddle and A. Malhotra. An $e \log e$ line crossing algorithm for levelled graphs. In J. Kratochvíl, editor, *Proc. Graph Drawing, GD 1999*, volume 1731 of *LNCS*, pages 59–70. Springer, 1999.

[143] J. N. Warfield. Crossing theory and hierarchy mapping. *IEEE Transactions on Systems, Man, and Cybernetics*, 7(7):502–523, 1977.

[144] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications (Structural Analysis in the Social Sciences)*. Cambridge University Press, 1994.

[145] V. Weber. Beziehungsgeflecht: Social-Network-Dienste zwischen Adressbuchersatz, Spielplatz und Kontakthof. *c't magazin für computer und technik*, 25:156–161, 2007.

[146] R. Wiese, M. Eiglsperger, and M. Kaufmann. yFiles: Visualization and automatic layout of graphs. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Proc. Graph Drawing, GD 2001*, volume 2265 of *LNCS*, pages 453–454. Springer, 2002.

[147] A. Yamaguchi and A. Sugimoto. An approximation algorithm for the two-layered graph drawing problem. In T. Asano, H. Imai, D. T. Lee, S.-i. Nakano, and T. Tokuyama, editors, *Proc. International Conference on Computing and Combinatorics, COCOON 1999*, volume 1627 of *LNCS*, pages 81–91. Springer, 1999.

[148] K.-P. Yee, D. Fisher, R. Dhamija, and M. Hearst. Animated exploration of dynamic graphs with radial layout. In *Proc. IEEE Symposium on Information Visualization, INFOVIS 2001*, pages 43–50, 2001.

Publications

- C. Bachmaier, U. Brandes, and F. Schreiber. Biological networks. In R. Tamassia, editor, *Handbook of Graph Drawing and Visualization*. CRC, 2010. To appear.
- C. Bachmaier, F. J. Brandenburg, W. Brunner, and F. Hübner. A global k-level crossing reduction algorithm. In M. S. Rahman and S. Fujita, editors, *Proc. Workshop on Algorithms and Computation, WALCOM 2010*, volume 5942 of *LNCS*, pages 70–81. Springer, 2010.
- C. Bachmaier, H. Buchner, M. Forster, and H. Seok-Hee. Crossing minimization in extended level drawings of graphs. *Discrete Applied Mathematics*, 158(3):159–179, 2010.
- C. Bachmaier, F. J. Brandenburg, W. Brunner, and R. Fülöp. Coordinate assignment for cyclic level graphs. In H. Q. Ngo, editor, *Proc. Computing and Combinatorics, COCOON 2009*, volume 5609 of *LNCS*, pages 66–75. Springer, 2009.
- C. Bachmaier, F. J. Brandenburg, W. Brunner, and G. Lovász. Cyclic leveling of directed graphs. In I. G. Tollis and M. Patrignani, editors, *Proc. Graph Drawing, GD 2008*, volume 5417 of *LNCS*, pages 348–349. Springer, 2009.
- C. Bachmaier, F. J. Brandenburg, W. Brunner, A. Hofmeier, M. Matzeder, and T. Unfried. Tree drawings on the hexagonal grid. In I. G. Tollis and M. Patrignani, editors, *Proc. Graph Drawing, GD 2008*, volume 5417 of *LNCS*, pages 372–383. Springer, 2009.
- C. Bachmaier and W. Brunner. Linear time planarity testing and embedding of strongly connected cyclic level graphs. In D. Halperin and K. Mehlhorn, editors, *Proc. European Symposium on Algorithms, ESA 2008*, volume 5193 of *LNCS*, pages 136–147. Springer, 2008.
- C. Bachmaier, W. Brunner, and C. König. Cyclic level planarity testing and embedding (extended abstract). In S.-H. Hong, T. Nishizeki, and W. Quan, editors, *Proc. Graph Drawing, GD 2007*, volume 4875 of *LNCS*, pages 50–61. Springer, 2007.
- C. Bachmaier. A radial adaption of the sugiyama framework for visualizing hierarchical information. *IEEE Transactions on Visualization and Computer Graphics*, 13(3):583–594, 2007.

- C. Bachmaier and M. Forster. Crossing reduction for hierarchical graphs with intra-level edges. Technical Report MIP-0608, University of Passau, 2006.

- C. Bachmaier and M. Forster. A radial adaption of the Sugiyama framework for hierarchical graph drawing. Technical Report MIP-0603, University of Passau, April 2006.

- C. Bachmaier, U. Brandes, and B. Schlieper. Drawing phylogenetic trees. In X. Deng and D. Du, editors, *Proc. Intl. Symposium on Algorithms and Computation, ISAAC 2005*, volume 3827 of *LNCS*, pages 1110–1121. Springer, 2005.

- C. Bachmaier, F. Fischer, and M. Forster. Radial coordinate assignment for level graphs. In L. Wang, editor, *Proc. Computing and Combinatorics, COCOON 2005*, volume 3595 of *LNCS*, pages 401–410. Springer, 2005.

- C. Bachmaier. Radiale Level-Planarität und -Einbettung in Linearzeit. In D. Wagner, editor, *Ausgezeichnete Informatikdissertationen*, volume D-5 of *GI-Edition Lecture Notes in Informatics*, pages 19–28. Gesellschaft für Informatik, 2005.

- C. Bachmaier, F. J. Brandenburg, and M. Forster. Radial level planarity testing and embedding in linear time. *Journal of Graph Algorithms and Applications*, 9(5):53–97, 2005.

- C. Bachmaier and M. Raitner. Improved symmetric lists. Technical Report MIP-0409, University of Passau, October 2004.

- C. Bachmaier, F. J. Brandenburg, M. Forster, P. Holleis, and M. Raitner. Gravisto: Graph visualization toolkit. In J. Pach, editor, *Proc. Graph Drawing, GD 2004*, volume 3383 of *LNCS*, pages 502–503. Springer, 2004.

- C. Bachmaier. *Circle Planarity of Level Graphs*. Dissertation, Faculty of Informatics and Mathematics, University of Passau, 2004.

- C. Bachmaier, F. J. Brandenburg, and M. Forster. Track planarity testing and embedding. In P. Van Emde Boas, J. Pokorný, M. Bieliková, and J. Štuller, editors, *Proc. Software Seminar: Theory and Practice of Informatics, SOFSEM 2004*, volume 2, pages 9–17. MatFyzPres, 2004.

- M. Forster and C. Bachmaier. Clustered level planarity. In P. Van Emde Boas, J. Pokorný, M. Bieliková, and J. Štuller, editors, *Proc. Software Seminar: Theory and Practice of Informatics, SOFSEM 2004*, volume 2932 of *LNCS*, pages 218–228. Springer, 2004.

- C. Bachmaier, F. J. Brandenburg, and M. Forster. Radial level planarity testing and embedding in linear time (extended abstract). In G. Liotta, editor, *Proc. Graph Drawing, GD 2003*, volume 2912 of *LNCS*, pages 393–405. Springer, 2004.

- C. Bachmaier, F. J. Brandenburg, and M. Forster. Radial level planarity testing and embedding in linear time. Technical Report MIP-0303, University of Passau, June 2003.
- C. Bachmaier. Vergleich und Implementierung von Partitionierungsverfahren auf Graphen. Diplomarbeit, Fakultät für Informatik und Mathematik, Universität Passau, 2000.

Index

Statutory Declaration

This habilitation thesis with the title “A Generalized Framework for Drawing Directed Graphs” is a result of my personal work and no other than the indicated aids have been used for its completion. All quotations and statements that have been inferred literally or in general manner are marked as such.

Passau, October 30, 2009

Christian Bachmaier