UNIVERSITÄT PASSAU

Fakultät für Informatik und Mathematik

Dissertation

Cyclic Level Drawings of Directed Graphs

Wolfgang Brunner

Supervisor
Prof. Dr. Franz J. Brandenburg

January 7, 2010

Dissertation for the aquisition of the degree of a doctor in natural sciences at the Faculty of Computer Science and Mathematics of the University of Passau.

1st referee: Prof. Dr. Franz J. Brandenburg, University of Passau
2nd referee: Prof. Dr. Ulrik Brandes, University of Konstanz

Abstract

The Sugiyama framework proposed in the seminal paper [140] of 1981 is one of the most important algorithms in graph drawing and is widely used for visualizing directed graphs. In its common version, it draws graphs hierarchically and, hence, maps the topological direction to a geometric direction. However, such a hierarchical layout is not possible if the graph contains cycles, which have to be destroyed in a preceding step. In certain application and problem settings, e. g., bio sciences or periodic scheduling problems, it is important that the cyclic structure of the input graph is preserved and clearly visible in drawings. Sugiyama et al. also suggested apart from the nowadays standard horizontal algorithm a cyclic version they called recurrent hierarchies. However, this cyclic drawing style has not received much attention since.

In this thesis we consider such cyclic drawings and investigate the Sugiyama framework for this new scenario. As our goal is to visualize cycles directly, the first phase of the Sugiyama framework, which is concerned with removing such cycles, can be neglected. The cyclic structure of the graph leads to new problems in the remaining phases, however, for which solutions are proposed in this thesis. The aim is a complete adaption of the Sugiyama framework for cyclic drawings.

To complement our adaption of the Sugiyama framework, we also treat the problem of cyclic level planarity and present a linear time cyclic level planarity testing and embedding algorithm for strongly connected graphs.

Preface

Writing such a thesis can never be done alone. Hence, it is my pleasure to express my gratitude to those who supported me over the last several years. First of all, I would like to thank Professor Dr. Franz J. Brandenburg for introducing me to the field of graph drawing. The corresponding lecture and my diploma thesis showed me that graph drawing is a very interesting field of research. Later, as a research assistant, he gave me the freedom to pursue my own ideas but always had the time to discuss related problems. He gave me the opportunity to attend several conferences which encouraged me to go on with my research. Thanks are also due to Dr. Christian Bachmaier who approached me with the problem of cyclic level planarity and got all this started. He was the main discussion partner for all problems in this thesis and was very helpful regarding technical problems. I wish to thank Professor Dr. Ulrik Brandes for co-supervising this thesis.

Thanks are appropriate for all further colleagues Christopher Auer, Andreas Gleißner, Andreas Hofmeier, Christof König, and Marco Matzeder who were part of several important discussions. I want to thank Christopher Auer especially for carefully proof reading the thesis. Furthermore, I am grateful to Gergö Lovász, Ferdinand Hübner, and Raymund Fülöp, who implemented parts of the cyclic Sugiyama framework in the scope of their diploma theses. I wish to thank Andreas Donig, Andreas Gleißner, and Kathrin Hanauer who helped to keep Gravisto alive. Last but not least, I am grateful to my friends and family for the support I received.

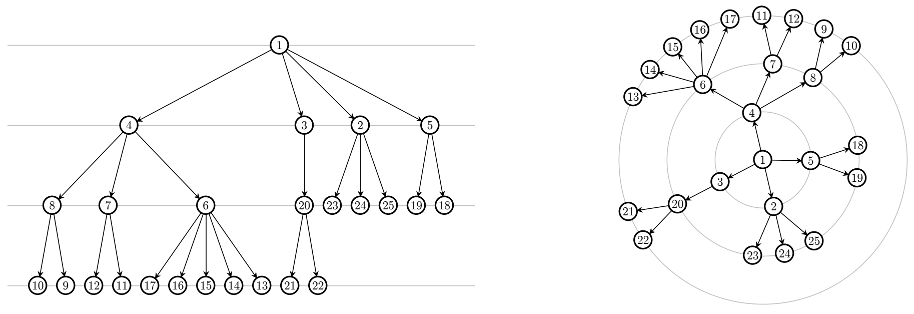
Wolfgang Brunner

Contents

1 Introduction

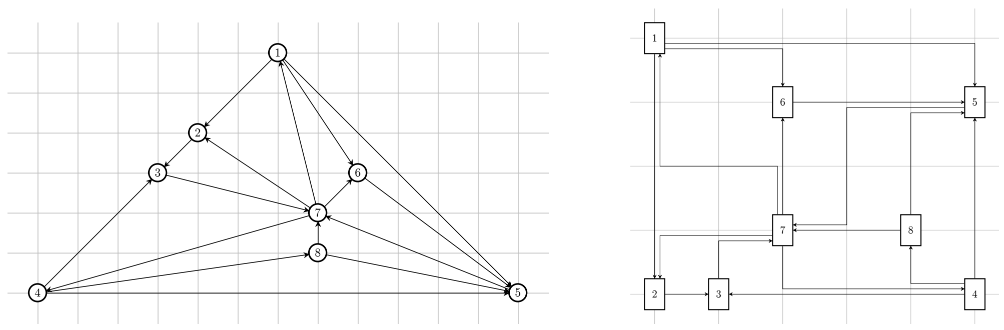
In computer science, graphs are used whenever arbitrary binary relations between entities are modeled. The field of application includes communication networks, entity-relationship diagrams, flow charts, street-maps, and circuit diagrams to name a few. Hence, graphs are a general-purpose tool. Common representations of graphs include storing the relations of the entities in a matrix, i. e., an adjacency matrix, or constructing a list of neighbors for each entity, i. e., an adjacency list. These representations are perfectly suited for the automatic processing in computers. For a human, however, it is very difficult to handle these representations although they contain the information completely. Remember the popular proverb saying “A picture is worth a thousand words”. A human can comprehend the information represented in a drawing of a graph much more easily than by having a look at, e. g., the adjacency matrix alone. In most drawings the entities are represented by circles or rectangles and the relations between them by lines connecting the entities. For small graphs, such a drawing can be made by hand. However, for graphs with several thousand entities these drawings have to be produced automatically. The field of research dealing with this problem is called *graph drawing*. For an overview on various graph drawing software see [90].

It is surprisingly hard to construct a “good” drawing of a graph. There is consensus that using small area, having few crossings or avoiding overlapping of the entities improves a drawing. Nevertheless, many different algorithms even for the same type of graphs have been proposed. For instance, trees can be drawn hierarchically (see Figure 1.1(a)) by the algorithm of Reingold and Tilford [126] and others [22, 26] and radially (see Figure 1.1(b)) by the algorithm of Eades [41]. For planar graphs several methods have been suggested as well, e. g., the algorithm by de Fraysseix, Pach, and Pollack [31] and others [91, 131] draw them on a grid with all edges as straight lines (see Figure 1.2(a)). Another possibility are orthogonal drawings [15, 142]. One such algorithm is the Kandinsky model [58] by Fößmeier and Kaufmann (see Figure 1.2(b)). For arbitrary graphs, i. e., force-based approaches [40, 64] (see Figure 1.3(a)) or high-dimensional embedding [75] (see Figure 1.3(b)) are widely used.



(a) Hierarchical drawing (b) Radial drawing

Figure 1.1. Example drawings of a tree

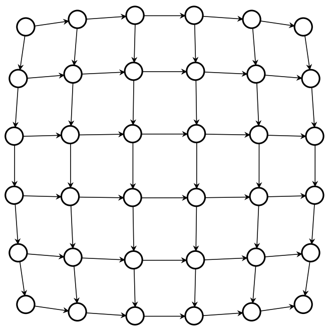


(a) Straight-line drawing (b) Orthogonal drawing

Figure 1.2. Example drawings of a planar graph

Each of these drawing algorithms has its own advantages and disadvantages and produces distinct drawings for the same graph. In Figure 1.2(a), e. g., a human can more easily realize which vertices are connected by edges in comparison to Figure 1.2(b). On the other hand, Figure 1.2(b) is better suited for circuit schematics or entity-relationship diagrams due to its orthogonal edge routing. Hence, often the choice which algorithm to use is heavily influenced by the concrete application setting. Therefore, it is important to have several alternatives for each type of graph. We present a new framework for drawing directed graphs in this thesis.

Many approaches to draw directed graphs are based on the Sugiyama framework [140], which is among the most intensively investigated algorithms in graph drawing. It is the standard technique to draw directed graphs, and displays them in an


(a) Force-based drawing

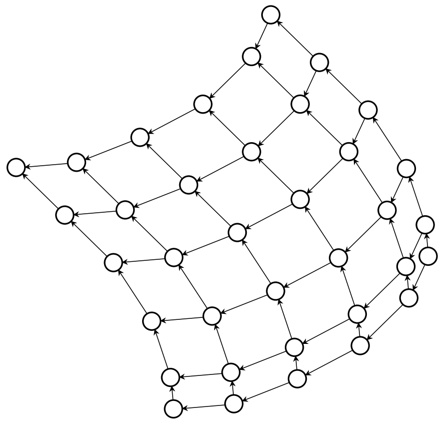

(b) Drawing constructed by high-dimensional embedding

Figure 1.3. Example drawings of a grid

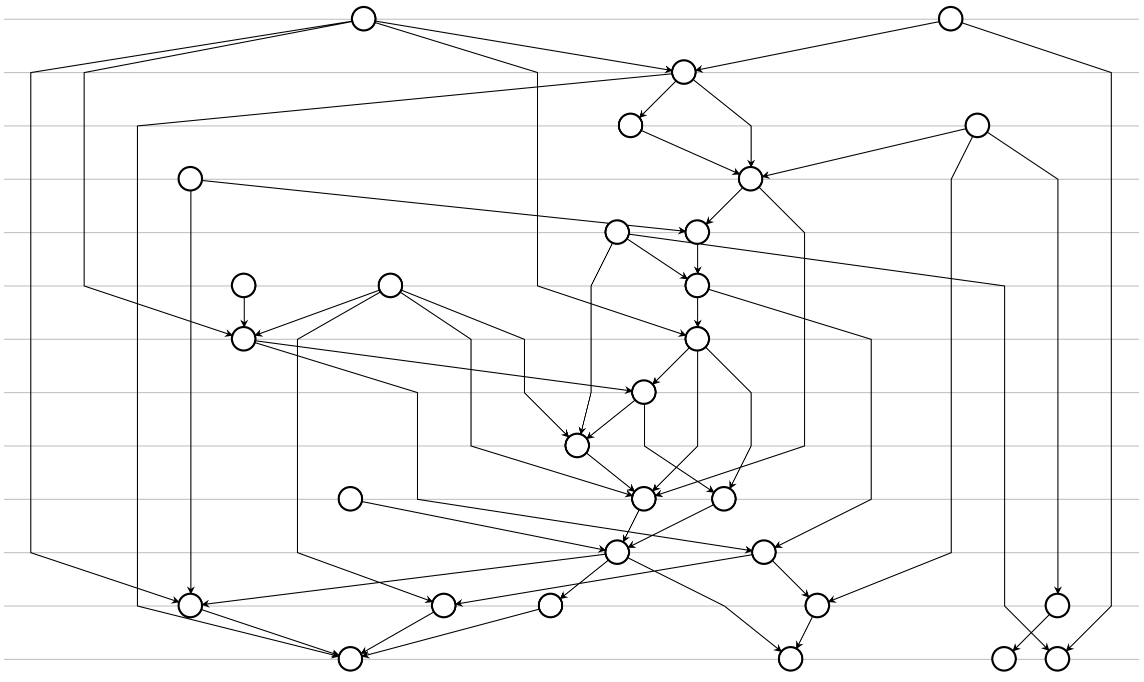


Figure 1.4. Drawing produced by the hierarchical Sugiyama framework

hierarchical manner. It is well-suited particularly for directed acyclic graphs, which are drawn top-down and level by level. These drawings reflect the underlying graph as a partial order. Typical applications include schedules, UML diagrams, and flow charts. See Figure 1.4 for an example of a typical Sugiyama drawing.

To draw an arbitrary directed graph hierarchically, the Sugiyama framework has to destroy cycles in a first step. It either removes or redirects edges until the resulting graph is acyclic. However, there are many situations, where this approach is unacceptable: In the bio sciences many chemical reactions are cyclic processes, i. e., the fatty acid synthesis (see Figure 1.5). It is a common standard there to display these cycles as such. These cycles often serve as a landmark [110] in larger drawings. While single metabolic paths may have a rather simple structure, representing the whole metabolism results in many cyclic dependencies. There exist algorithms by Schreiber [132] to draw such structures. However, to visualize the cyclic parts in the drawings in a symmetrical way, these are made by hand in general.

A second application are periodic scheduling problems. Often daily, weekly or monthly activities repeat periodically. The assembly line in a factory, for instance, starts each day exactly where it left the day before. Cyclic drawings visualize such an overnight dependency in the same way as each dependency during the day. In public transportation, periodic schedules are used for planes, trains, or busses. Figure 1.6 represents the daily schedule of trains operating between Paris and Lyon in the 1880s [147]. There, the x-axis denotes the time from 6 a. m. to 6 a. m. the next day and the y-axis denotes the position between Paris (top) and Lyon (bottom). Each line in the diagram represents a train operating between the two cities. Following a train from 5 a. m. to 7 a. m. in the drawing is rather difficult as the drawing is cut at 6 a. m. In [147] Tufte suggests to mount the diagram on a cylinder to avoid this problem. Variants of these scheduling problems called the periodic event scheduling problem [134] or the cyclic job-shop problem [74] have been studied in the literature.

In the design of micro chips often groups of several parts, e. g., transistors, have to be placed in a regular pattern on a long strip [107]. One goal is to keep the area needed on the chip as small as possible. Minimizing the area for one group does not solve this problem in general. Survey the simple example in Figure 1.7(a). The group consisting of three parts is drawn with minimal width assuming all connections are to point to the right. However, not considering each group separately results in a more narrow drawing (see Figure 1.7(b). This is only possible if an algorithm capitalizes on the periodic structure of the circuit. As modern micro chips use billions of transistors manual layout is not possible.

Figure 1.8 shows a deterministic finite automaton (DFA) [81] using the alphabet $\Sigma = \{a, b\}$. It accepts all words with a length being a multiple of six and having an even number of a's in it. Note that nearly all DFAs have cycles as acyclic DFAs describe a finite language. Similar visualizations can be used for, e. g., Petri nets [118], Markov chains [108], or (recurrent) neural networks [105] where cycles are common as well.

Simple cyclic drawings consisting of only one cycle are very common. They are called cycle diagrams and are often used to visualize cyclic processes. In Figure 1.9

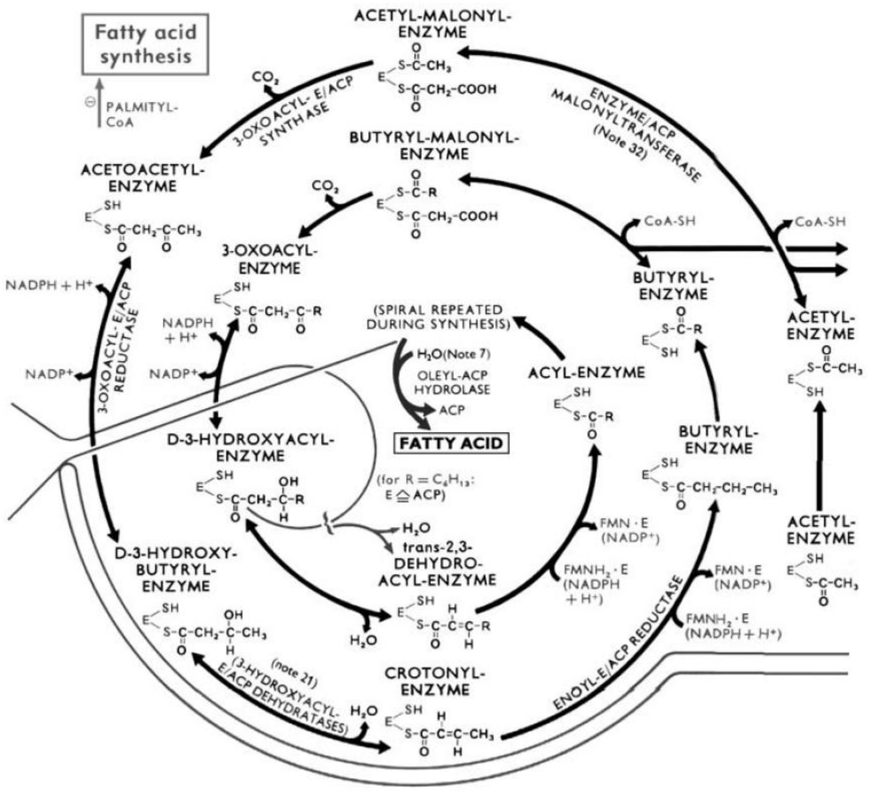


Figure 1.5. Fatty acid synthesis [109]

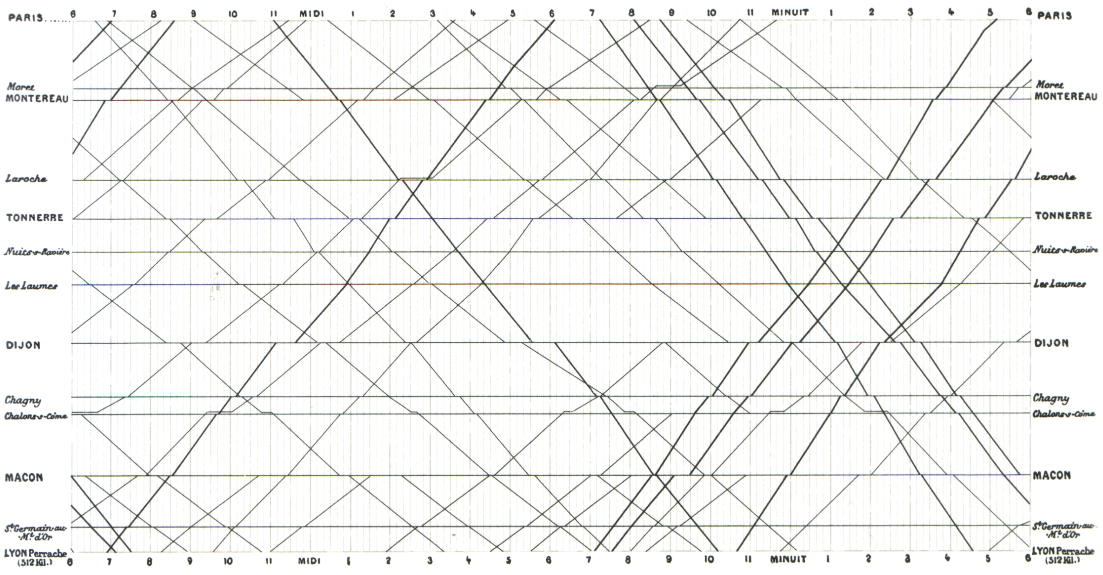
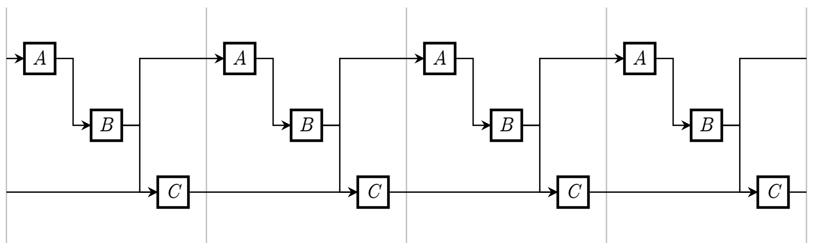
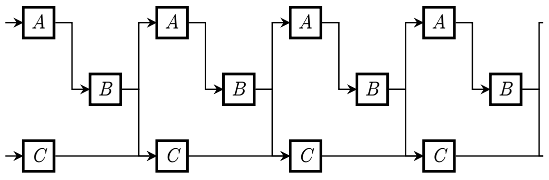


Figure 1.6. Daily schedule of trains between Paris and Lyon in the 1880s [147]



(a) Large width



(b) Smaller width

Figure 1.7. Different area consumption in integrated circuit layout [107]

the working capital cycle [67] is shown. Such drawings are often used in presentations as, e. g., Microsoft PowerPoint can generate them easily. However, to generate more complicated cyclic diagrams, more sophisticated algorithms are needed.

In this thesis we adapt and extend the Sugiyama framework to make it applicable for cyclic layouts of graphs. We call this new version *cyclic Suyigama framework* and call the traditional version the *hierarchical Sugiyama framework*. In Figure 1.10(a) the graph G^* is drawn with the hierarchical Sugiyama framework and in Figure 1.10(b) our new cyclic variant is used. We will use the graph G^* as an continuous example in the remainder of this thesis. The resulting drawings of G^* of each phase are handmade to be able to illustrate problems in the various phases more easily.

The thesis is organized as follows: In the following section we give basic definitions needed in this thesis. After that, we are able to describe the hierarchical Sugiyama framework in more detail in Chapter 1.3. We discuss its shortcomings and give an overview over the cyclic version we propose. Each of the four phases of the Sugiyama framework and its extension proposed by this thesis for cyclic drawings is examined in a separate chapter. These are the phases decycling (Chapter 2), leveling (Chapter 3), crossing reduction (Chapter 4) and coordinate assignment (Chapter 5). The special case of level planarity, i. e., having no edge crossings at all in a level drawing, is treated in Chapter 6. In Chapter 7 we compare drawings produced by the hierarchical and the cyclic Sugiyama framework empirically. Finally, Chapter 8 gives a summary and open problems.

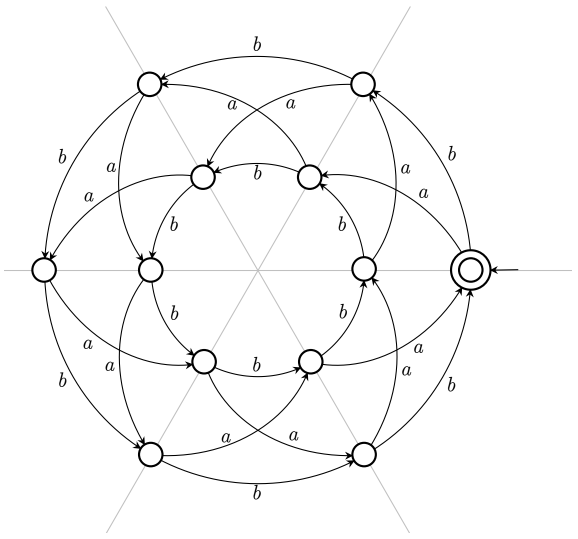


Figure 1.8. Finite automaton

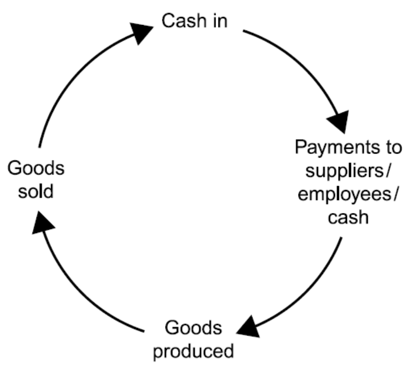
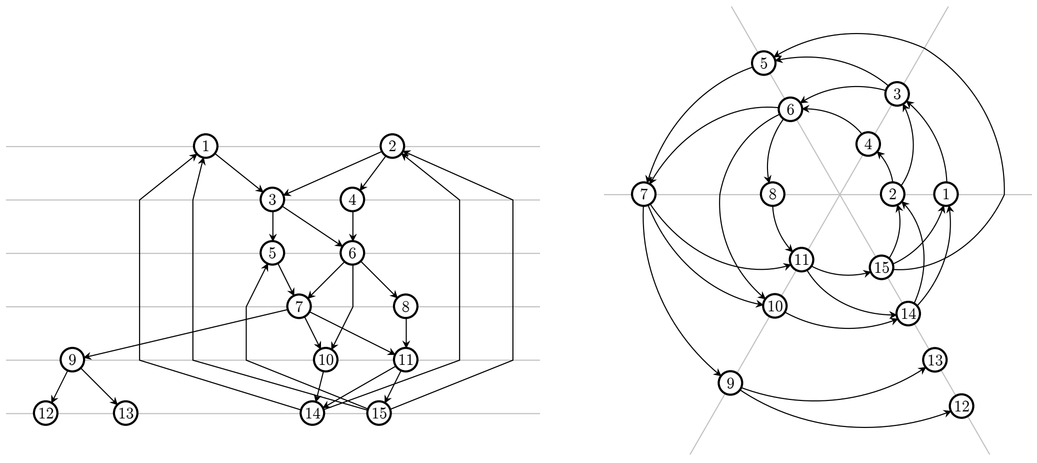


Figure 1.9. Working capital cycle [67]



(a) Hierarchical drawing

(b) Cyclic drawing

Figure 1.10. Drawings of the example graph G^*

1.1 Preliminaries

This chapter gives basic definitions needed in this thesis. We adopt the definitions of common textbooks for graph drawing [32, 93, 137].

1.1.1 Graph

A *directed graph* $G = (V, E)$ consists of a finite set of *vertices* V and a finite set of *edges* $E \subseteq V \times V$. For each edge $e = (u, v)$ u is called the *start vertex* of e and v is called the *end vertex* of e. In this case e is an *outgoing* edge of u and *incoming* edge of v. The vertices u and v are *adjacent* or *neighbors*. They are both *incident* to the edge e and vice versa. The *adjacency list* of a vertex v contains all vertices adjacent to v. If the direction of an edge is ignored both incident vertices are called end vertices. The number of incoming edges of a vertex v is denoted with $\deg^-(v)$ and is called the *in-degree* of v. Accordingly, the number of outgoing edges is denoted with $\deg^+(v)$ and is called *out-degree*. The *degree* $\deg(v)$ of a vertex v is the sum of its in-degree and out-degree. The degree of a graph is the maximum degree of its vertices. A vertex without incoming edges is called *source*, a vertex without outgoing edges is called *sink*, and a vertex without indicent edges is *isolated*. *Reversing* an edge (u, v) results in the edge (v, u). A *self-loop* in a graph is an edge starting and ending at the same vertex.

In a graph $G = (V, E)$, a *path* is a sequence of vertices $v_1, \dots, v_k$ such that $\forall i \in \{1, \dots k-1\} : (v_i, v_{i+1}) \in E$. A *way* is a sequence of vertices $v_1, \dots, v_k$ such that $\forall i \in \{1, \dots k-1\} : (v_i, v_{i+1}) \in E \vee (v_{i+1}, v_i) \in E$. Informally, in a path the direction of each edge is respected whereas in a way the direction is ignored. A path or way is *simple* if the vertices $v_1, \dots v_k$ are mutually unequal. Two paths or ways are *edge disjoint* if the sets of used edges are disjoint. An edge $e = (u, v)$ is *transitive* if there is a path from u to v in $G = (V, E \setminus \{e\})$. If $v_1 = v_k$, then a path $v_1, \dots, v_k$ is called a *cycle* and such a way $v_1, \dots, v_k$ is called a *circle*. A cycle or circle is *simple* if the vertices $v_1, \dots v_{k-1}$ are mutually unequal. A directed graph without cycles is a *directed acyclic graph (DAG)*.

Two vertices v_1 and v_2 are *connected* if there exists a way between v_1 and v_2. These vertices are *strongly connected* if there exist paths from v_1 to v_2 and from v_2 to v_1. A graph is *connected* or *strongly connected* if each pair of vertices is connected or strongly connected, respectively. A graph $G' = (V', E')$ is a subgraph of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E \cap (V' \times V')$. In the context of several graphs, we denote with $V(G')$ the set of vertices and with $E(G')$ the set of edges of a graph G'. A connected (strongly connected) subgraph of G which is maximal regarding inclusion is a *connected (strongly connected) component* of G.

An *undirected graph* $G = (V, E)$ consists of a finite set of *vertices* V and a finite set of *edges* $E \subset \mathcal{P}(V)$ consisting of subsets of V with exactly one or two elements. An edge $e = \{u\}$ is called a *self-loop* on vertex u. For each edge $e = \{u, v\}$, u and v are the *end vertices* of e. The vertices u and v are called *adjacent* or *neighbors*. The vertices u and v are *incident* to e and vice versa. The degree $\deg(v)$

of v denotes the number of incident edges of v. Ways, and circles, connectivity, connected components and subgraphs are defined analogously for undirected graphs. The *underlying undirected graph* $G' = (V', E')$ of a directed graph $G = (V, E)$ is defined by $V' = V$ and $E' = \{\{u, v\} \in \mathcal{P}(V) | (u, v) \in E\}$. An undirected graph $G = (V, E)$ is *complete* if for each pair of vertices $u, v \in V$ $(u \neq v)$ the edge $\{u, v\} \in E$ exists. G is *bipartite* if the set of vertices V can be partitioned into $V = V_1 \dot\cup V_2$ such that for each edge $e = \{u, v\}$ one end vertex is part of V_1 and the other is part of V_2. G is *complete bipartite* if it is bipartite with $V = V_1 \dot\cup V_2$ and each vertex of V_1 is adjacent to each vertex of V_2.

Two undirected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are *isomorphic* if there exists a bijection $f : V_1 \to V_2$ such that for each pair of vertices $u, v \in V_1$ $\{u, v\} \in E_1$ if and only if $\{f(u), f(v)\} \in E_2$. An undirected graph H is a *subdivision* of an undirected graph G, if H can be constructed by repeatedly replacing an edge $e = \{u, v\}$ of G by two edges $\{u, w\}$ and $\{w, v\}$ using a new vertex w. Two undirected graphs G_1 and G_2 are *homeomorphic* if isomorphic subdivisions of G_1 and G_2 exist.

A connected directed graph $G = (V, E)$ is a *directed tree* if it contains exactly one vertex r with no incoming edge and all other vertices have in-degree one. The vertex r is called the *root* of the tree. For an edge (u, v) u is called the *parent* of v and v is the *child* of u. A vertex without an outgoing edge is called *leaf*. The vertex v is a successor of u if a path from u to v in G exists.

For simple algorithms on graphs like *depth first search* (DFS), *breadth first search* (BFS), or *topological sorting* we refer to [30].

1.1.2 Level Graph

Let $G = (V, E)$ be a directed acyclic graph. For a given number of levels $k \in \mathbb{N}$ $(k \geq 2)$ we call a function $\phi \colon V \to \{1, \ldots, k\}$ a *level assignment* of G if for each edge $e = (u, v) \in E$ $\phi(u) < \phi(v)$ holds. We call $G = (V, E, \phi)$ a *k-level graph* or *level graph* for short, if the number of levels k is clear by context or unimportant. For two vertices $u, v \in V$, let $\text{span}(u, v) = \phi(v) - \phi(u)$.

Let $G = (V, E)$ be a directed graph. For a given number of levels $k \in \mathbb{N}$ $(k \geq 1)$, we call the function $\phi \colon V \to \{1, 2, \ldots, k\}$ a *cyclic level assignment* of G and $G = (V, E, \phi)$ a *cyclic k-level graph* or *cyclic level graph* for short. Note that $\phi(u) \geq \phi(v)$ for an edge $e = (u, v)$ is possible in the cyclic case and that each k-level graph is a cyclic k-level graph as well. For two vertices $u, v \in V$ 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.

Let $G = (V, E, \phi)$ be a (cyclic) k-level graph. We call each $l \in \{1, \ldots, k\}$ a *level*. For an edge $e = (u, v) \in E$, we define $\text{span}(e) = \text{span}(u, v)$. Note that in the cyclic case an edge $e = (u, v)$ with $\phi(u) = \phi(v)$ has $\text{span}(e) = k$. An edge e is *short* if $\text{span}(e) = 1$ and *long* otherwise. A graph is *proper* if all edges are short. Each (cyclic) level graph can be made proper by adding $\text{span}(e) - 1$ *dummy vertices* for each edge e which split e in $\text{span}(e)$ many short edges, called *segments*. A segment connecting an original vertex and a dummy vertex is called an *outer segment* and a segment between two dummy vertices is an *inner segment*. For a (cyclic) k-level graph $G = (V, E, \phi)$ we denote with $G^p = (V^p, E^p, \phi^p)$ the proper version of G. Hence, V consists of all original vertices and V^p additionally contains all dummy vertices. Furthermore, E is the set of all original edges whereas E^p contains the segments of G^p only. The leveling ϕ^p additionally assigns each dummy vertex its level.

Let $V_i = \{v \in V \mid \phi(v) = i\}$ be the set of vertices on level i. We define the *width* of a leveling as $\max_{1 \leq i \leq k} |V_i|$, i. e., the maximum number of (original) vertices on a level, and the *height* of a leveling as the number of levels k. Note that a (cyclic) leveling ϕ is surjective in most cases to avoid empty levels. The adjacency list of each vertex v of a (cyclic) level graph is split into two parts where $N^-(v)$ denotes the vertices connected to v by an incoming edge and $N^+(v)$ denotes the vertices connected by an outgoing edge.

A proper (cyclic) k-level graph $G^p = (V^p, E^p, \phi^p, <)$ is *ordered* or *embedded* if $<$ is a total ordering for each V_i^p $(1 \leq i \leq k)$. Two segments (u, v), (w, x) starting at the same level *cross* if $u < w$ and $v > x$ or $u > w$ and $v < x$. The ordered proper (cyclic) k-level graph G^p is *(cyclic) level planar* if is has no crossings. Two segments in G^p starting at the same level have a *conflict* if they cross or share a vertex. Conflicts are of *type 0, 1* or *2*, if none, one or two of the conflicting segments are inner segments, respectively. In an ordered level graph the adjacency lists $N^-(v)$ and $N^+(v)$ of each vertex v are ordered by $<$.

1.2 Drawings of Graphs

There are many ways to draw a graph, e. g., inclusion diagrams [45] or visibility representations [141]. In both cases, edges are not represented directly. Inclusion diagram are used to visualize trees. Vertices are represented by rectangles and one vertex v is an successor of a vertex u if the rectangle of v is drawn inside the rectangle of u. Tsiaras et al. [146] generalized this idea do draw directed acyclic graphs as DAG-maps. Visibility representations are used for planar graphs. Vertices are represented by horizontal lines and two vertices are connected if there exists a vertical line crossing both representations of the two vertices which does not cross any other horizontal line.

However, the most common drawing style in the plane, which we also use in this thesis, is as follows: A *drawing* $D = (d_v, d_e)$ consists of an injective function $d_v : V \to \mathbb{R}^2$ and a function $d_e : E \to (\mathbb{R}^2)^{[0,1]}$. The function d_v assigns each vertex v a point p_v with *x-coordinate* $x(v)$ and *y-coordinate* $y(v)$ in the plane. The function d_e assigns each edge $e = (u, v)$ a curve with starting point $d_e(u)(0) = p_u$ and end point $d_e(v)(1) = p_v$. We visualize a vertex as a small circle with the center p_v and visualize an edge by drawing a line along its curve which is clipped at the circles of its start and end vertex. We add a small arrow at the end of the clipped curve belonging to the end vertex if the edge is directed. Depending on the application, different line styles, colors, or thicknesses for the vertices or edges are used. In the remainder of this thesis, we abstract from these graphical details and do not distinguish between a drawing D and its visual representation. In all drawings in the plane we use a coordinate system with the x-axis pointing to the right and the y-axis pointing downwards. We call a drawing of an edge *straight line* if the curve is an affine linear function. A drawing of an edge is a *polyline* if its curve is piecewise affine linear. The non-differentiable points in the curve are called *bends*. Two edges *cross* in a drawing if their curves have a common point which is not a common end point. A drawing is *plane* if it does not contain a crossing. A graph G is *planar* if a plane drawing of G is possible. The *width* and *height* of a drawing is the difference of the maximal and minimal x-coordinates and y-coordinates used by a vertex or edge in a drawing, respectively. The *area* is the product of width and height of a drawing. A *face* of a plane drawing is a contiguous area of the drawing bounded only by the lines representing edges. Sorting the adjacency lists of each vertex v of a planar graph G corresponding to the counter-clockwise order of the incident edges of v gives a *planar embedding* of G. The *dual graph* G' of a planar graph G with a fixed embedding consists of a vertex for each face of G. Two vertices in G' are adjacent if the corresponding faces in G are bounded by a common edge.

We also consider drawings of graphs on three dimensional objects like cylinders or tori. As in the two dimensional case, we assign vertices to points and edges to curves on the surface of these objects.

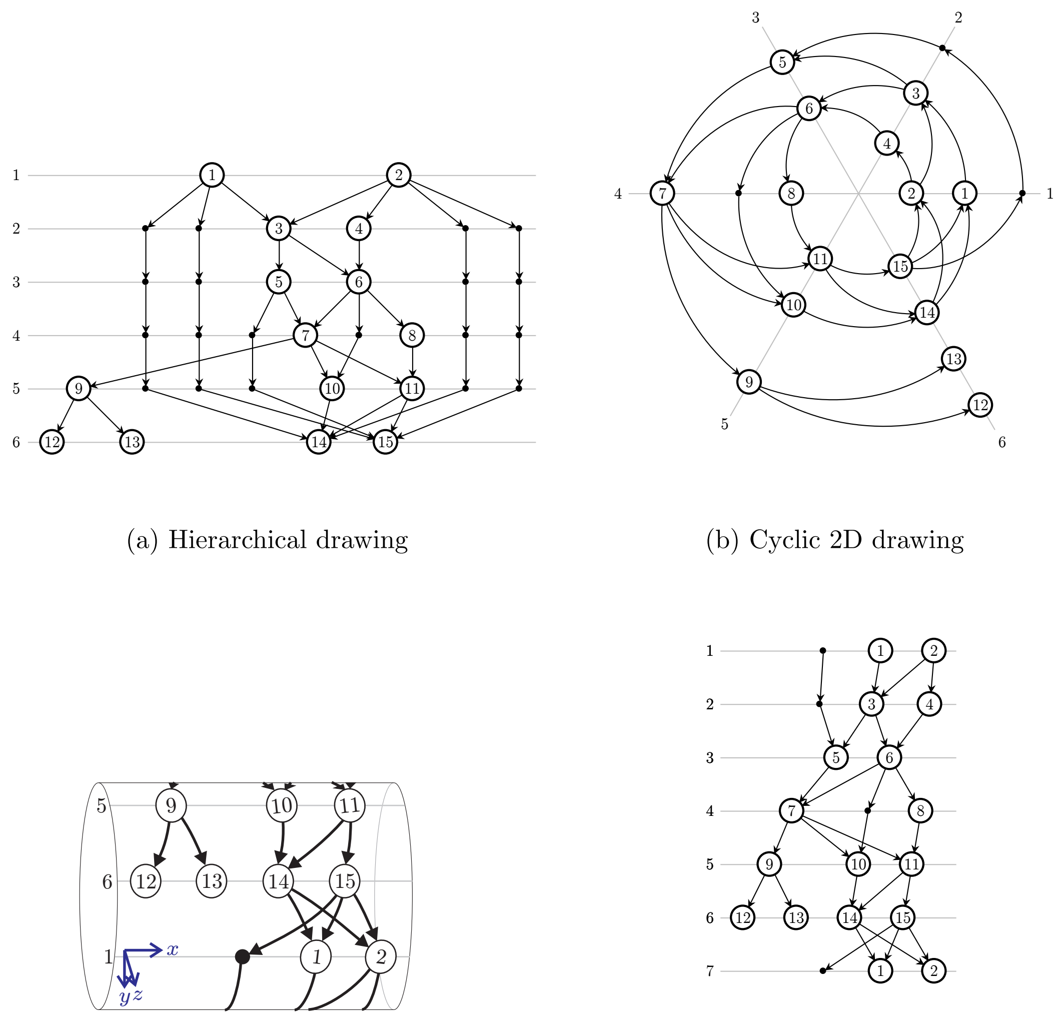
A *drawing of a k-level graph* $G = (V, E, \phi)$ or *hierarchical drawing* (see Figure 1.11(a)) is determined only by the coordinates of all (dummy) vertices in its proper version $G^p = (V^p, E^p, \phi^p)$ as we draw each segment with a straight line between its end points. We place all (dummy) vertices of V_i^p on a horizontal line called *level line* such that V_{i+1}^p lies below V^p. Hence, all edges point downwards. As the y-axis points downwards as well, using $y(v) = \phi^p(v)$ for each vertex $v \in V$ gives correct y-coordinates. If G^p is ordered, we draw the vertices in V_i from left to right on each level, i. e., for two vertices $u, v \in V_i$ $x(u) < x(v)$ if $u < v$. We represent dummy vertices by small black circles in drawings.

A cyclic k-level graph $G = (V, E, \phi)$ can be drawn in different ways, e. g., in a *2D drawing*, where the level lines form a star and each edge is directed counter-clockwise (see Figure 1.11(b)) or in a *3D drawing*, where the level lines lie on the surface of a cylinder and all edges wrap around the cylinder in the same direction (see Figure 1.11(c)). We call both drawings *cyclic drawings*, and they can easily be transformed into each other. We represent drawings of cyclic level graphs in an *intermediate drawing* of the proper version $G^p = (V^p, E^p, \phi^p)$ by assigning each (dummy) vertex $v \in V^p$ coordinates $x(v) \in \mathbb{R}$ and $y(v) = \phi^p(v) \in \mathbb{N}$ (see Figure 1.11(d)). All vertices of V^p on level 1 are duplicated and put on a new 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* $\frac{1}{x(v)-x(u)}$. Again, if G^p is ordered $x(u) < x(v)$ if $u < v$ holds for each pair of vertices $u, v \in V_i^p$ on the same level i.

A 2D drawing as in Figure 1.11(b) is obtained from an intermediate drawing by transforming each point $p = (x(p), y(p))$ of the plane to $\big(x_{2\text{D}}(p), y_{2\text{D}}(p)\big) = \big(r(p) \cdot \cos(\alpha(p)), -r(p) \cdot \sin(\alpha(p))\big)$, with the radius $r(p) = (\text{offset}_x + \max_{v \in V}(x(v))) - x(p) \cdot \delta_x$

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 Figure 1.11(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. For the 3D drawing we use a left-handed coordinate system with its point of origin on the axis of the cylinder and the x-axis pointing along the axis of the cylinder. For the orientation of the coordinate system see Figure 1.11(c).


(a) Hierarchical drawing

(b) Cyclic 2D drawing

(c) Cyclic 3D drawing

(d) Intermediate drawing

Figure 1.11. Drawings of the example graph G^* with dummy vertices

These equations transform straight lines of the intermediate drawing to spiral segments in the 2D or 3D drawings. Due to these different representations, we define, analogously to level graphs, that if we traverse an edge it its direction we traverse *downwards* or *counter-clockwise*. We use both notations interchangeably depending on which of them is more appropriate in the respective context. Traversing an edge against its direction is called traversing *upwards* or *clockwise*. Traversing the vertices $v_1 < v_2 < \ldots < v_{|V_i^p|}$ of a level i of an ordered proper (cyclic) level graph $G^p = (V^p, E^p, \phi^p, <)$ is called traversing the level from *left* to *right*. This corresponds to traversing a level *towards the center* in the 2D drawing. The x-coordinate of a vertex corresponds to its *radius* in polar coordinates in the 2D drawing and the y-coordinate corresponds to the *angle*.

A (cyclic) level drawing is *plane* if it does not contain a crossing. A (cyclic) level graph G is *(cyclic) level planar* if a plane drawing of the (cyclic) level graph G is possible.

In the drawings in Figure 1.11 the level numbers are shown. We omit them wherever possible and use the convention that in the cyclic 2D drawing the level 1 is directed to the right.

1.2.1 Aesthetic Criteria

The decision, whether or not a drawing is nice, always depends on the intended use and is rather subjective. A drawing can be perfect for one case and be rather unsuitable for another one, e. g., for circuit schematics an orthogonal drawing like in Figure 1.2(b) suites much better than a drawing as displayed in Figure 1.2(a). Nevertheless, it is important to have some general *aesthetic criteria* which allow us to evaluate the quality of a drawing by means of numerical values. This enables to compare drawings formally. Then, for each application scenario, the emphasis can be put on suitable aesthetic criteria. Commonly used aesthetic criteria [10, 123, 140] include:

- Area

  The goal is to keep the area of the drawing small. Variants are to confine only the width or height of a drawing. Another possibility is to address the aspect ratio, which is the ratio of the width and the height. Here, a drawing with aspect ratio 1 or using the golden ratio of $(1+\sqrt{5})/2$ is preferable over, i. e., a wide and low drawing using less area.

- Crossings

  Keeping the number of crossings small improves the readability of a drawing. At best, planar graphs allow for a drawing without crossings at all. If avoiding crossings is impossible, trying to maximize the angle of a crossing is an option.

- Edge length

  Shorter edges can be understood more easily. Therefore, the sum of all edge lengths or the maximum edge length are kept small. A variant is to try to give all edges nearly the same edge length.

- Bends

  Bends on edges interfere with the readability of the drawing. Ideally, each edge is drawn as a straight line. If bends cannot be avoided, the goal is to keep the total number or the maximum number of bends per edge small.

- Angle

  Two edges incident to the same vertex having a small angle between them are hard to distinguish. Thus, maximizing the smallest angle in a drawing improves its readability.

- Symmetry

  Identical or similar structures in a graph should be represented in the same or a similar way in a drawing.

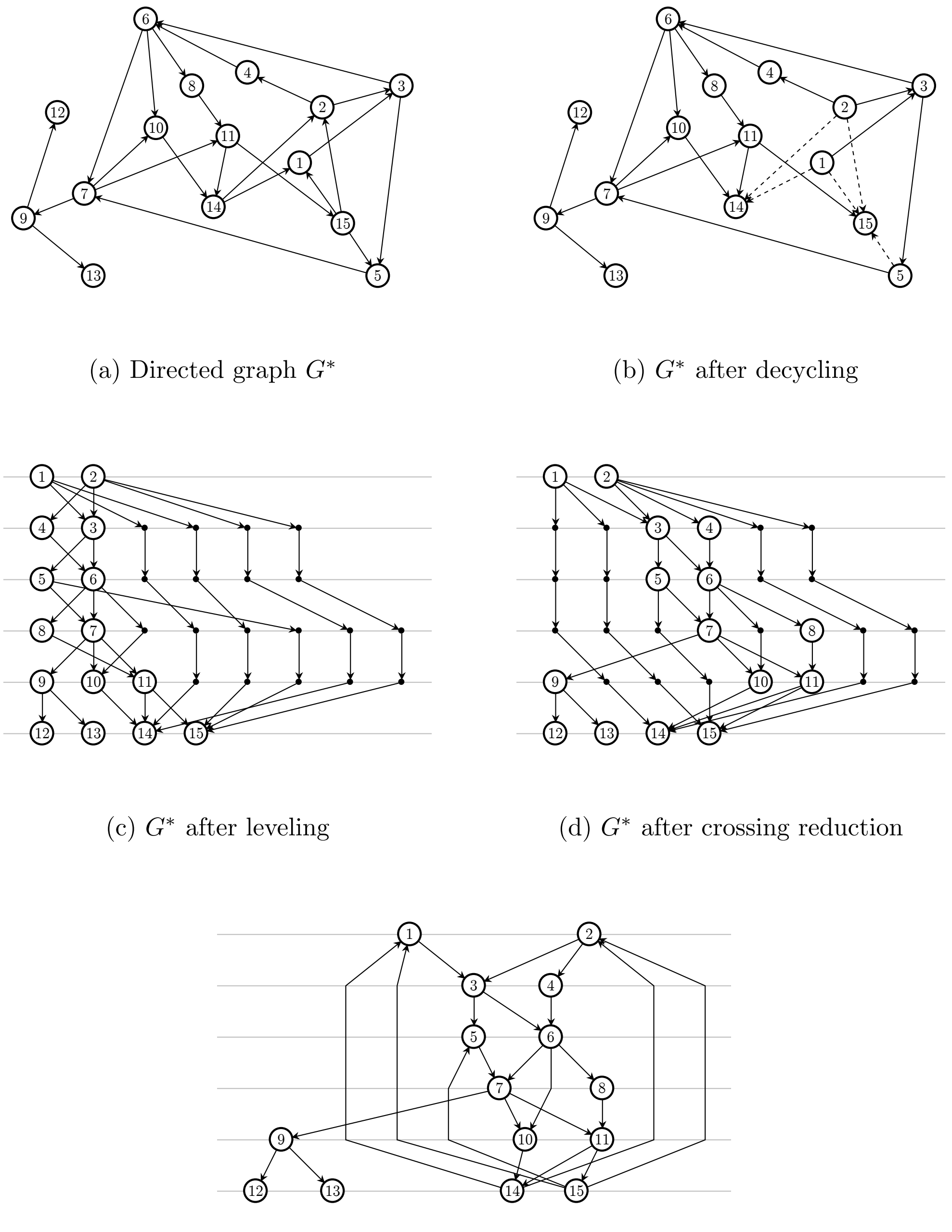
These criteria have been evaluated empirically by Purchase et al. [121, 122, 151]. Especially crossings increase the visual complexity and hinder humans to easily comprehend a drawing. In general it is an $\mathcal{NP}$-hard problem to construct drawings that minimize or maximize any of these aesthetic criteria. On the other hand, optimizing one criterion and ignoring all other may not be a good idea. Minimizing, e. g., the area may lead to a drawing with many bends. On the other hand, drawing without bends may lead to exponential area in some cases. Hence, a good heuristic has to find a proper trade-off between all aesthetic criteria.

1.3 Hierarchical Sugiyama Framework

In their seminal paper of 1981, Sugiyama et al. [140] suggested a new drawing algorithm for directed graphs. It was well accepted as the Sugiyama framework and is now the standard algorithm to draw directed graphs. It is especially well suited for acyclic graphs and displays them in a hierarchical manner. The framework consists of the four phases decycling, leveling, crossing reduction and coordinate assignment.

See Figure 1.12 for an example. If the given input graph (see Figure 1.12(a) which uses the example graph G^*) is not acyclic then the first phase removes all cycles by reversing the direction of a subset of all edges (see Figure 1.12(b)). As reversing the edges changes the graph itself, the goal is to find a small subset of the edges to reverse. Finding the minimum set is the $\mathcal{NP}$-hard feedback arc set problem [69, 92]. Hence, heuristics are used to address the problem.

The now acyclic graph serves as input to the second phase called leveling. Here each vertex v is assigned a natural number $\phi(v)$ which is called its level (see Figure 1.12(c)). These are used as y-coordinates in the final drawing. The leveling is chosen such that each edge is directed from a smaller to a higher level. Edges spanning several levels are split into shorter edges spanning only one level by inserting dummy vertices on these edges. The goals are to keep the number of levels and the number of vertices on each level small and to spread the vertices evenly on the



(a) Directed graph G^*

(b) G^* after decycling

(c) G^* after leveling

(d) G^* after crossing reduction

(e) G^* after coordinate assignment

Figure 1.12. Example of the hierarchical Sugiyama framework

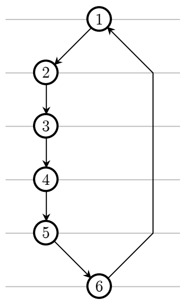
levels. The former two goals correspond to the $\mathcal{NP}$-hard precedence constrained scheduling problem [69, 148]. Again, heuristics have to be used.

The crossing reduction is the third phase of the framework. It permutes the vertices on its levels to reduce the overall number of edge crossings (see Figure 1.12(d)). This problem is traditionally approached by utilizing the one-sided 2-level crossing reduction problem in several up and down sweeps. But even this problem, where only the crossings between two neighboring levels are examined and the permutation of one level is fixed, is $\mathcal{NP}$-hard [44].

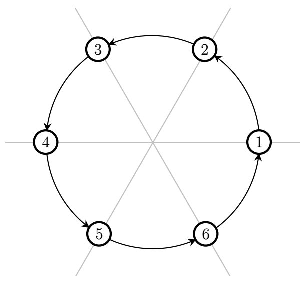
The final phase is called coordinate assignment, where the x-coordinate of each vertex is determined. To achieve a nice drawing, the goals are to avoid bends on edges where possible, to align long edges vertically, and to center each vertex between its neighbors. As a simple postprocessing, the introduced dummy vertices are removed and replaced by bends where necessary and the reversed edges are put in their original direction (see Figure 1.12(e)). A more detailed description of each of the four phases is given in the next chapters.

1.3.1 Drawbacks of the Hierarchical Sugiyama Framework

Although the Sugiyama framework is one of the standard algorithms to draw directed graphs, there are some drawbacks. It destroys all cycles and redirects some edges. These run from a higher to a lower level, tend to be long and may cause many crossings. Therefore these edges appear to be special in the drawing although they are probably not in the original graph. Consider a simple cycle as in Figure 1.13. At least one edge has to be reversed to draw the cycle in the hierarchical Sugiyama framework (see Figure 1.13(a)). This randomly chosen edge is long, points upwards and dominates the drawing. In a larger drawing, several such cycles exist which result in several long reversed edges. A much more natural representation of the cycle is shown in Figure 1.13(b), where all edges are drawn in the same way.



(a) Hierarchical



(b) Cyclic

Figure 1.13. Drawings of a simple cycle

1.4 Cyclic Sugiyama Framework

In their original paper from 1981 [140], Sugiyama et al. propose a solution for 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, at least two drawings seem natural: The first one is a 2D drawing, where the level lines are rays from a common center, and are sorted counter-clockwise by their number (see Figure 1.14(d)). 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 from u to v wrapping around the center at most once. The second is a 3D drawing, where the levels are on the surface of a cylinder parallel to its axis and all edges wrap around the cylinder in the same direction (see Figure 1.15(a)).

A combination of the two drawing methods could combine the advantage of having a 2D drawing and using parallel level lines instead of rays. This can be realized by utilizing an interactive 2D view. This view can be scrolled upwards and downwards infinitely and always shows a different part of the cylinder like Figure 1.15(b). Another example is a version of Figure 1.6 scrollable infinitely to the left and right.

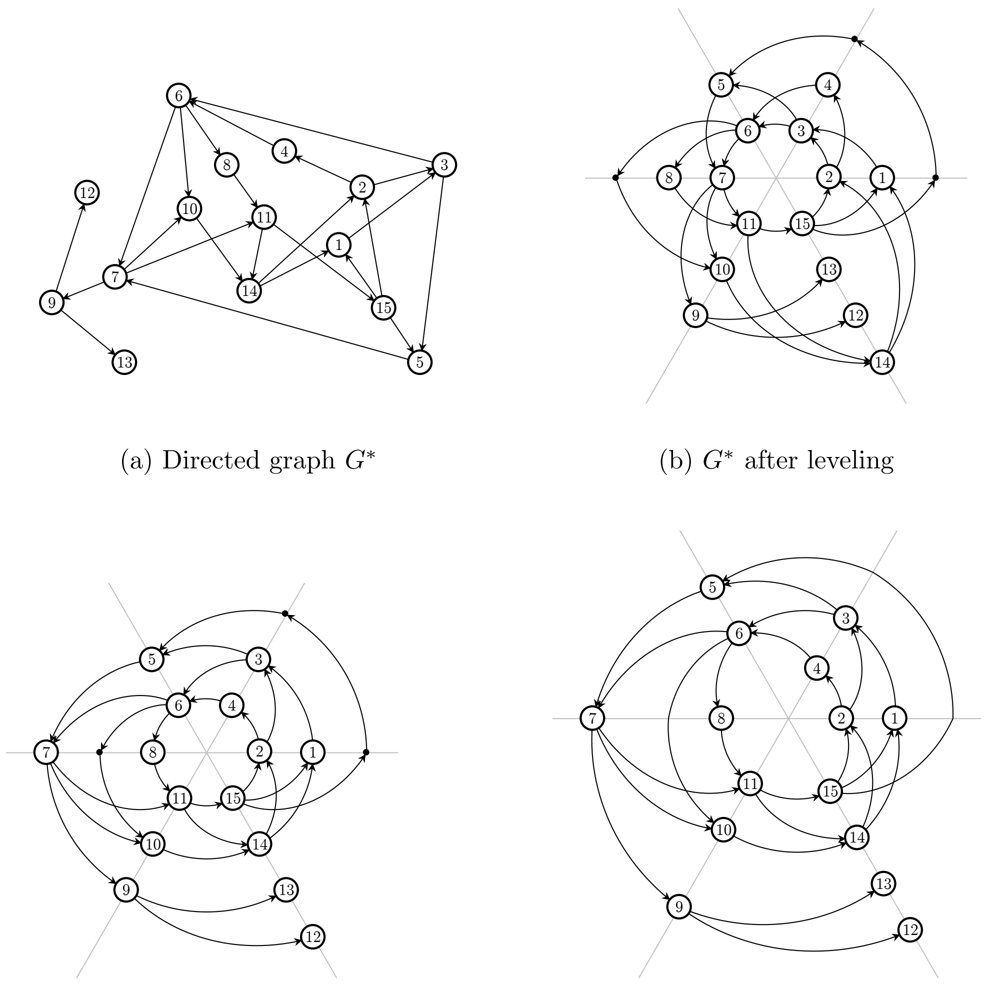
Recurrent hierarchies are known to most graph drawers – but unnoticed. A planar recurrent hierarchy is shown on the cover of the book by Kaufmann and Wagner [93]. There, Bastert and Matuszewski state that recurrent hierarchies are "unfortunately [. . .] still not well studied". One reason is that the problems encountered with cyclic drawings a more complicated in comparison to the standard hierarchical case. Intuitively, there are no first and last levels, there are (possibly) no sources or sinks where algorithms may start to process the input graph. So, the algorithms for each phase have to use a more global view for each problem.

In cyclic drawings, edges are irreversible and cycles are represented in a direct way. Thus, the cycle removal phase is not needed any more. This saves much effort, since the underlying problem is the $\mathcal{NP}$-hard feedback arc set problem [69]. Another advantage are short edges, as the sum of the edge length can be smaller than in the hierarchical case: Consider a simple example of a cycle C consisting of three vertices. The only way to draw C in the Sugiyama framework is to reverse one edge which then spans two levels. Therefore, the sum of the edge lengths is at least four. In the cyclic case this graph can be drawn on three levels such that each edge has span one. Moreover, the cyclic style can reduce the number of crossings. See Figure 1.12(e) and Figure 1.14(d) as an example.

The leveling, crossing reduction, and coordinate assignment phases remain but new problems which arise due to the cyclic structure of the graph have to be solved. To avoid confusion, we will still number the remaining phases from phase two to phase four.

Given an input graph (see Figure 1.14(a) which uses the example graph G^*) the leveling phase assigns a level to each vertex. This level can now be interpreted as the angle of the vertex in its polar coordinates (see Figure 1.14(b)). The crossing reduction permutes the vertices on their level (see Figure 1.14(c)) as in the hierarchical case. Finally, the coordinate assignment phase fixes the radius of each vertex (see Figure 1.14(d)).

In order to reflect an inherent symmetry of a graph in its drawing, one goal of the cyclic framework is to treat all levels equally, i. e., no phase uses a special treatment for edges from the last to the first level.


(a) Directed graph G^*

(b) G^* after leveling

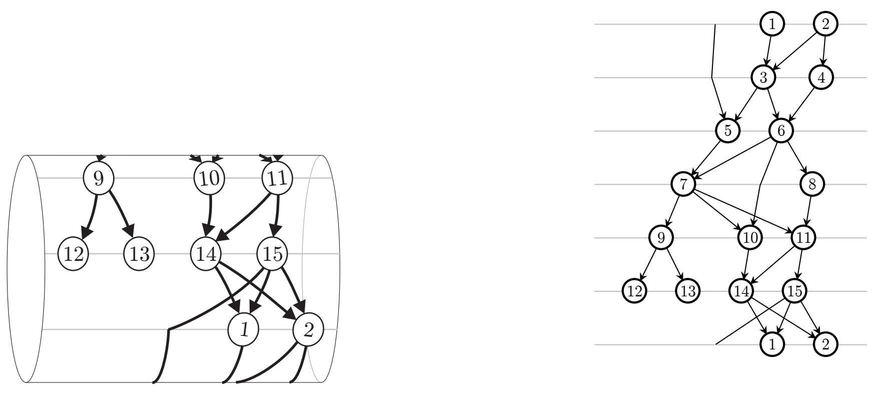
(c) G^* after crossing reduction

(d) G^* after coordinate assignment

Figure 1.14. Example of the cyclic Sugiyama framework

1.4.1 Why to extend the Sugiyama Framework?

The Sugiyama framework is the standard algorithm to draw directed graphs. One reason for its popularity is that it produces visually appealing drawings. Also its modular design of splitting the task of drawing a graph into four distinct phases might have helped the Sugiyama framework to gain its popularity. By exploiting



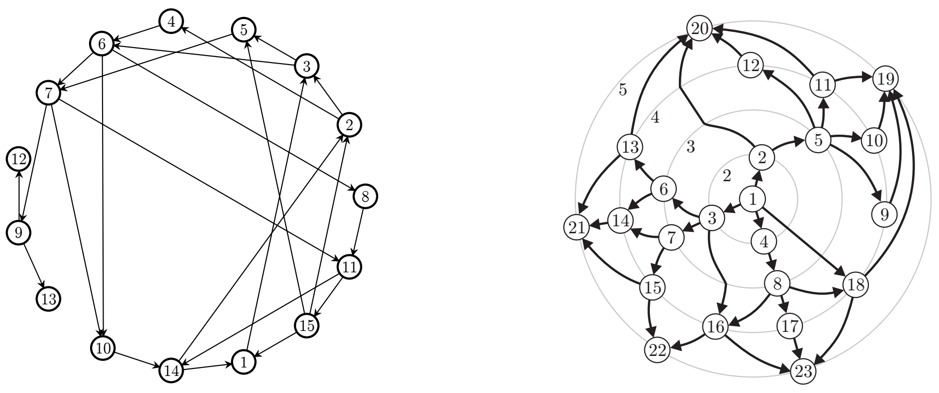
(a) G^* drawn on a cylinder (b) Cylinder cut open

Figure 1.15. Variants of the drawing of G^* of Figure 1.14(d)

this modularity, for each phase an algorithm can be provided independently from the other phases, which most suitable solves the problem at hand. For each phase rather simple, easy to implement and more sophisticated algorithms exist. Thus, the framework can be adapted to a concrete application scenario by taking into account the trade-off between simplicity and quality of the used algorithms for each phase. The separation into distinct phases is also applicable for the cyclic version. Sugiyama et al. already had the idea in their seminal paper [140] to use recurrent hierarchies. Nevertheless, there are alternatives which we discuss here.

Sugiyama [137] mentions two completely different alternatives to the hierarchical and the cyclic framework to draw cyclic directed graphs. The first alternative computes the strongly connected components of the graph and then each component is shrunk to a single vertex. The result is an directed acyclic graph which can be drawn, e. g., with the hierarchical Sugiyama framework. The original graph is drawn in 3D using a plane for each level. For each vertex of the DAG the corresponding strongly connected component is drawn on the corresponding plane then. There are several problems with this approach: It is only suitable for graphs with small strongly connected components, as the approach does not tell how to draw these components. Hence, if the graph is one large strongly connected component, the original problem still exists. Additionally, most applications need a 2D drawing. However, both problems can be tackled with the cyclic Sugiyama framework. This framework can be used to draw each strongly connected component in the same plane as the remaining graph. The result is a combination of the hierarchical and the cyclic framework (see Figure 8.5).

The second approach mentioned by Sugiyama is to use algorithms for undirected graphs like spring embedder [40, 64] or high-dimensional embedding [75]. In [137] it is stated that this is only possible “as long as the expression of the flow is not important”. In this thesis, the goal is to visualize cycles directly, so the direction of the edges is fundamental. Hence, this approach was not investigated further. Another approach to draw undirected graphs is to put all vertices on a circle and draw all edges as straight lines. The main goal here is to minimize the number of crossings [11]. But, the cycles in the graph are not clearly visible as the edge direction is ignored again. See Figure 1.16(a) for an example of a circular drawing of the example graph G^*.



(a) Circular drawing of G^* (b) Radial drawing of an acyclic graph

Figure 1.16. Different drawing styles

Another approach is displayed in Figure 1.15(b). Here, some vertices are split such that one copy of such a vertex gets all its incoming edges and the other all its outgoing edges. The graph is acyclic then. But to find a suitable leveling for such a graph and to find the right vertices to split, a cyclic leveling is needed. To ensure that the permutations of the first and last level are the same and that the x-coordinates of the two copies of these vertices are the same, cyclic crossing reductions and coordinate assignment heuristics are needed. Hence, to get such drawings the cyclic Sugiyama framework is needed.

Apart from algorithms that produce completely different drawings than the Sugiyama framework, there are several other approaches that result in drawings similar to the Sugiyama framework. In [23], solving one-dimensional optimization problems for each axis is used to create the drawings. In [39], quadratic programming and force-based layout are combined to achieve hierarchical drawings. Both can visualize cyclic graphs without decycling them first but still give a mostly hierarchical drawing in general.

Even algorithms to produce cyclic drawings exist. Sugiyama and Misue [139] use the magnetic spring model, which is an extension of force-directed layouts, to produce cyclic layouts amongst others. However, like in most force-based layouts, properties like the general orientation of edges cannot be proven. Pich [119] proposes to use eigenvalues of a matrix associated with the input graph to create cyclic drawings. There, edges are drawn clockwise and counter-clockwise and can even cross vertices. Tamassia and Tollis [143] construct visibility representations of graphs on a cylinder which are only applicable to a subset of planar graphs.

Drawing graphs radially on levels forming concentric circles and edges pointing outwards is another possibility to visualize directed graphs [24, 125]. Although such drawings look similar to cyclic drawings at first glance, they are only suited for acyclic graphs. A radial version of the Sugiyama framework has been proposed by Bachmaier recently [2]. See Figure 1.16(b) for a radial level drawing.

In conclusion, there are distinct approaches but each of them has its drawbacks. In the end, the Sugiyama framework is the standard algorithm to draw directed graphs. The cyclic version has been suggested nearly 30 years ago but has not been treated yet. This thesis narrows this gap.

1.4.2 When to use which framework?

When comparing the resulting drawings of the example graph G^* in the hierarchical case (see Figure 1.12(e)) and the cyclic case (see Figure 1.14(d)) the cyclic drawing shows its benefits. The cyclic structure of the graph is clearly visible. The edges have similar lengths. In the hierarchical case, five edges have been reversed. Four of these edges are strikingly long and span from the last to the first level. The 24 edges have a span of 43 in total whereas the cyclic version has a span of 27 only. Long edges potentially cause more crossings as well. The hierarchical drawing has eleven crossings and the cyclic version has only four. Another benefit of cyclic drawings is that all edges are headed counter-clockwise. Consequently, visually understanding the drawing becomes easier. In fact, the arrow heads are not needed at all, as the direction of each edge is implicit.

But of course the example graph G^* is chosen to present the benefits of the cyclic Sugiyama framework. There are many graphs which are better suited for the hierarchical case, e. g., most acyclic graphs. The cyclic Sugiyama framework has its own drawbacks as well. One is that each cycle wraps around the center at least once. Hence, a short cycle will result in long edges. Furthermore, there are differences between the different drawings styles in the cyclic case. The 2D drawing uses curved lines that may be more difficult to comprehend for a human than straight lines. The edges in the 3D drawing are more easily to follow. However, in most application scenarios 2D drawings are preferred. If a graph has to be drawn on a computer screen, an interactive view using the surface of a cylinder gives the best results. Here, the drawing is 2D, all edges point downwards and are drawn as polylines, and the view can be scrolled upwards and downwards infinitely without encountering the end of the drawing.

The cyclic Sugiyama framework is not a replacement but an alternative to the existing hierarchical framework. Hence, the question arises when to use which framework. Obviously, an universal answer to that question is rather difficult, as this largely depends on the input graph and the application at hand.

A first attempt could be to suggest to draw acyclic graphs with the hierarchical framework and cyclic graphs which have many cycles of the same length with the cyclic framework. But Figure 1.5 shows that even acyclic or nearly acyclic graphs can benefit from the cyclic version. The input graph consists of a long path with some additional edges. Using the hierarchical framework would lead to a very high and narrow drawing and, consequently, a bad aspect ratio. The cyclic drawing gives an aspect ratio of one and allows the additional edges to be short. But apart from acyclic graphs with such special structures, drawings of acyclic graphs benefit from a hierarchical layout in general. Especially, when such special structures are not known beforehand.

Graphs with many cycles of the same length are well suited for the cyclic framework. Sugiyama himself stated in [137] that the cyclic version "can be considered effective for graphs with a large number of cycles". Moreover, the length of these cycles can be used as the number of levels to keep the span of the edges small. Unfortunately, although there are many results regarding the length and number of cycles in directed and undirected graphs [1, 13, 73, 84, 145], finding the average length of cycles in a graph seems to be a hard problem. However, if the semantic of the input graph is known, the context can give some hints. Cyclic scheduling problems are instances where cycles of the same length are to be expected. Drawing a daily schedule with 24 levels or a weekly schedule with seven levels with the cyclic framework is a natural choice.

If no additional information about a cyclic graph is given, no general recommendation about the method to use is possible. If only one particular graph has to be drawn, a simple approach is to construct both drawings and to compare them visually to decide which drawing reflects the requirements best. When drawing a larger number of graphs of similar structure, this decision could be made by investigating the drawings of only a small subset of the graphs. However, in the end, this will remain a matter of personal taste.

2 Decycling

This chapter describes the decycling phase of the Sugiyama framework. First, we summarize common techniques for the hierarchical case as shown in textbooks about graph drawing [32, 93, 137].

2.1 Hierarchical Sugiyama Framework

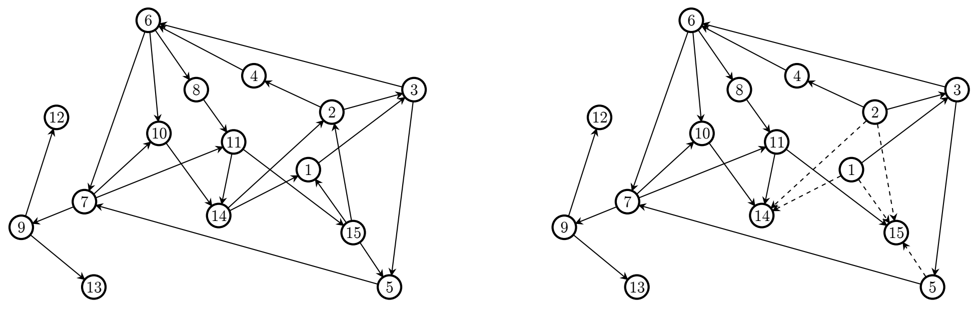
In the hierarchical Sugiyama framework, the input for the decycling phase is an arbitrary directed graph and the output is a directed acyclic graph. There are three possibilities to achieve this: The first is to shrink the strongly connected components to single vertices. The second is to remove and the third is to reverse edges. In all cases, these changes are reversed after the fourth phase. If some edges or strongly connected components were removed then these are not processed in the phases two through four. Thus, reinserting them is not a straight forward task. If the edges were reversed only, then it is easy to reverse them once more to return them to their original state. Hence, reversing edges is the only technique to use in practice. Figure 2.1(a) shows the directed graph G^* and Figure 2.1(b) shows an acyclic version of G^* with the five dashed edges reversed.

As changing the graph distorts the original information it contains, the goal is to reverse a set of edges which is as small as possible. This problem is the $\mathcal{NP}$-hard Feedback Arc Set Problem [69, 92]. Consequently, several heuristics are used. Here, we assume that the graph does not contain cycles of length two.

The problem of finding a small subset of edges to reverse parallels the problem of finding a total ordering of the vertices with a small number of edges going from a larger to a smaller vertex, see Figure 2.2. Choosing an arbitrary ordering or an ordering found by depth or breadth first search gives very poor results. In all three cases it is possible that nearly all edges are reversed. However, building an arbitrary ordering and taking that one or its reversed one depending on which gives the better result removes at most half of the edges.

The greedy cycle removal [47] repeats the following steps until the graph $G = (V, E)$ is empty and each vertex has been given a position in the total ordering of all vertices which is represented by an array of length $|V|$. As long as there are sources or sinks place them at the first or last free position of the array, respectively, and remove them from the graph. If the graph is not empty choose a vertex v with $\deg^+(v) - \deg^-(v)$ maximal and place it at the first free position and remove it.

Treating the strongly connected components [144] of a graph separately can improve the quality of the result considerably [46, 130] as edges between different strongly connected components never have to be reversed. There are, e. g., randomized algorithms [12] and exact approaches using branch and cut [72], as well.



(a) Directed graph G^*

(b) Acyclic version of G^* with reversed edges dashed

Figure 2.1. Hierarchical decycling

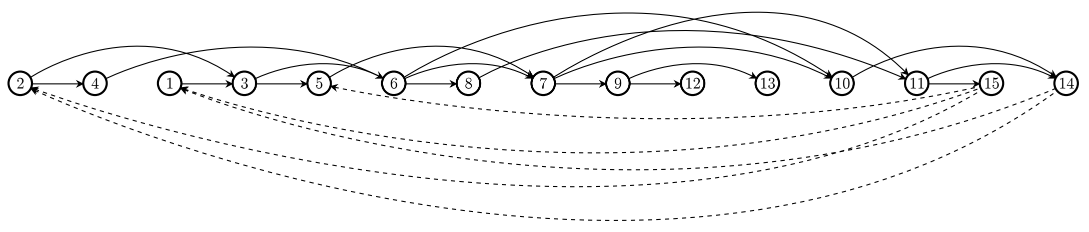


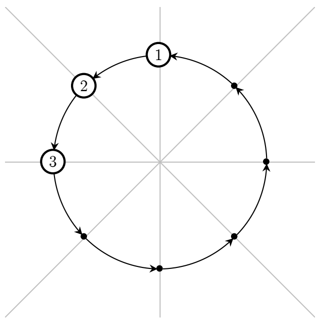
Figure 2.2. Ordering of G^* with the edges to reverse pointing from right to left

2.2 Cyclic Sugiyama Framework

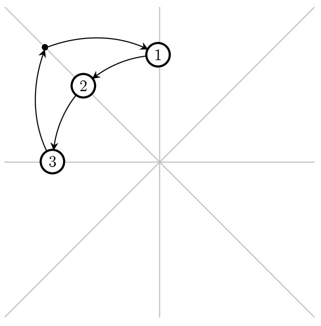
In the cyclic case, the direct representation of cycles is the main goal of the framework. The decycling phase is not needed and all cycles are untouched. Consequently, at least this $\mathcal{NP}$-hard problem is eliminated from the cyclic Sugiyama framework. But there is another huge benefit: All edges have the same counter-clockwise direction in the cyclic drawing. Hence, visually understanding the drawing is much easier. Actually, the arrow head at the end of each edge is not needed anymore as the direction is given implicitly. In the hierarchical case, the direction of each edge has to be comprehended separately. Even more, reversed edges in the hierarchical framework tend to be rather long and cause many crossings. Reversing an edge in a cycle gives the arbitrarily chosen edge a special appearance in the final drawing although the edge does not have any special properties in the original graph.

2.3 Open Problems

The decycling phase is not needed in the cyclic Sugiyama framework. However, we briefly discuss the option to reverse some edges to balance the lengths of all remaining cycles in the graph. This length can be used as the number of levels in the second phase. However, these reversed edges are oriented clockwise then. This is a heavy price to pay, as the general direction of the edges is disturbed, which is one of the main drawbacks of the hierarchical Sugiyama framework in the first place. Nevertheless, a short cycle in a drawing of a larger graph on many levels can benefit from a reversed edge. Figure 2.3 sketches this by showing the short cycle only. Figure 2.4 shows the same for a long cycle, where even a crossing can be avoided if clockwise edges are allowed. In both cases the span of the cycle is reduced by reversing one edge. Further research is needed to determine if reversing edges is sensible at all in the cyclic case.

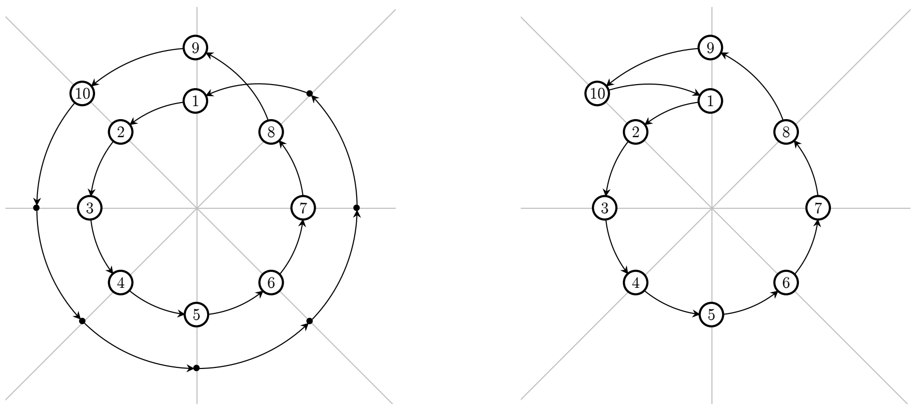


(a) Long non-reversed edge



(b) Short reversed edge

Figure 2.3. Two drawings of a short cycle



(a) Long non-reversed edge

(b) Short reversed edge

Figure 2.4. Two drawings of a long cycle

3 Leveling

In the leveling phase, each vertex is assigned a level which serves as a y-coordinate in the hierarchical case and as an angle in the cyclic case. An important goal is to keep the drawing compact, i. e., keep the number of levels small and spread the vertices evenly over the levels such that the maximum number of vertices per level stays small as well. Another goal is to keep the sum of the span of all edges or the maximum span of an edge small.

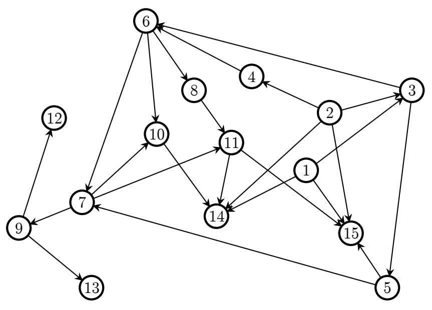
In the hierarchical case (see Figure 3.1), the leveling has to take the edge directions into account: For each edge $e = (u, v)$, the level of the start vertex u has to be smaller than the level of the end vertex v. In the cyclic case (see Figure 3.2), this is not needed: For an edge $e = (u, v)$ each leveling of u and v is feasible as the edge is drawn from $\phi(u)$ counter-clockwise until it reaches $\phi(v)$.

In some applications the leveling is already given, e. g., when drawing a schedule, where each task has a fixed point in time already. Here, e. g., using seven levels is suitable for a weekly schedule. If the number of levels is not given beforehand, an appropriate number of levels can be found during the leveling phase in the hierarchical case. In the cyclic case, however, the number of levels has to be determined beforehand. As new levels have to be added between existing ones, this may increase the span of already leveled edges. In the hierarchical case, adding new levels at the top or bottom is always possible without increasing the span.

In the hierarchical case, many leveling algorithms begin by placing some of the source or sink vertices on an extreme level. When using the cyclic Sugiyama framework many graphs to draw are cyclic or even strongly connected and do not have any sources or sinks. Hence, completely new leveling algorithms have to be found.

This chapter is organized as follows: In Section 3.1 we give some definitions. In Section 3.2 we compare the complexity of the hierarchical and the cyclic leveling and give $\mathcal{NP}$-hardness results. We present our cyclic leveling heuristics in Section 3.3

and give experimental results in Section 3.4. Most hierarchical leveling algorithms ignore the width of dummy vertices. We treat the problem of leveling with wide dummy vertices in the cyclic case in Section 3.5. Finally, we give a summary in Section 3.6 and state open problems in Section 3.7.


(a) Directed acyclic graph G of G^*

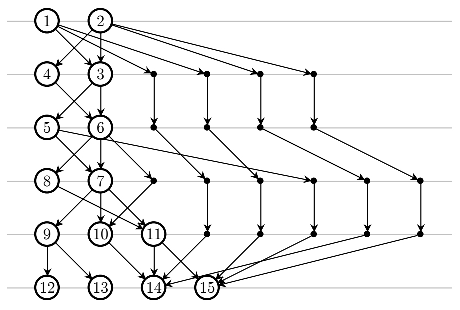
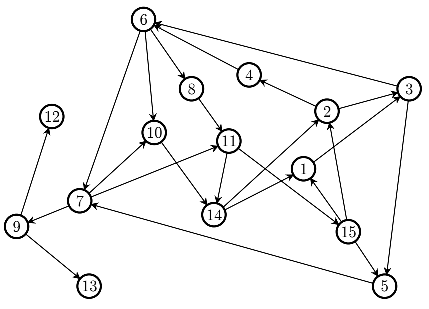

(b) Hierarchical leveling of G

Figure 3.1. Hierarchical leveling


(a) Cyclic graph G^*

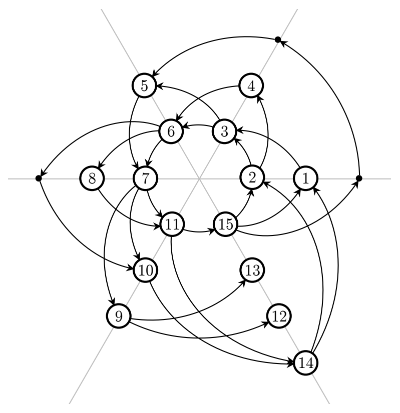

(b) Cyclic leveling of G^*

Figure 3.2. Cyclic leveling

3.1 Preliminaries

Let $G = (V, E, \phi)$ be a cyclic level graph. We define $\text{span}(G) = \sum_{e \in E} \text{span}(e)$. 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 below the level l. For a vertex $v \in V$ and a subset $V' \subset V$ we set $E(v, V') = \{\, (u, v) \in E \mid u \in V' \,\} \cup \{\, (v, w) \in E \mid w \in V' \,\}$.

3.2 Comparison of the Hierarchical and Cyclic Sugiyama Framework

In this section, we present different leveling problems and compare their complexity in the hierarchical and the cyclic case. For the shown hierarchical leveling algorithms, we follow common textbooks on graph drawing [32, 93, 137]. The graphs $G = (V, E)$ are directed in both cases and are acyclic in the hierarchical setting.

3.2.1 Dimensions of the Leveling

We examine three leveling problems which consider only the resulting height or width of the leveling. Note that dummy vertices are ignored when computing the width here. We treat their width as well in Chapter 3.5.

Problem 1. *Let $0 < k \in \mathbb{N}$. Does there exist a leveling of G with height at most k?*

In the hierarchical case the longest path heuristic can be used to get a leveling with minimal height which solves Problem 1. The heuristic places all sources on the first level and removes them temporarily from the graph. All new sources are placed on the second level and so on. Thus, for each vertex, the length of the longest path from a source determines its level. The number of levels needed is then the number of vertices in the longest path in the input graph which is obviously optimal. Because of its simplicity the longest path heuristic is often used.

In the cyclic case Problem 1 is trivial: Note that an edge $e = (u, v)$ does not impose any constraint on the leveling of the vertices u and v. The vertex u can have a smaller level, a larger level, and even the same level as v. Hence, it is always possible to place all vertices on one level.

Problem 2. *Let $0 < \omega \in \mathbb{N}$. Does there exist a leveling of G with width at most ω?*

This problem is trivial in both the hierarchical and cyclic case. In the hierarchical framework, a leveling of width one can be achieved by placing each vertex on its own level according to a topological sorting [30] of G. However, the resulting leveling has a bad aspect ratio. In the cyclic version any injective leveling can be used.

Problem 3. *Let $k, \omega \in \mathbb{N}$ $(k, \omega > 0)$. Does there exist a leveling of G with height at most k and width at most ω?*

Problem 3 is $\mathcal{NP}$-hard as this corresponds to the precedence constrained scheduling problem [69, 92, 148], where the vertices are the tasks to be executed, the edges are the precedence constraints, the width of the leveling is the number of processors, and the height is the deadline when all tasks have to be processed. Consequently, heuristics have to be used. Interestingly, the most common one is an algorithm for the precedence constrained scheduling problem by Coffman and Graham [29]. It assumes the input graph to have no transitive edges. These can be removed by a preprocessing step in linear time [106]. Note that this does not affect the width of the leveling. The removed edges are reinserted after the leveling.

The algorithm by Coffman and Graham operates in two phases. It uses a lexicographical ordering of sets of integers, where the largest number in each set is compared first, then the second largest number and so on. First the vertices are labeled iteratively from 1 to $|V|$ where an arbitrary source is labeled with 1. The k-th label is given to the unlabeled vertex whose predecessors have all been labeled and which has the minimal set of labels of predecessors using the ordering described above. Then the levels are filled from bottom to top respecting the given width ω. Levels are filled with vertices whose successors have already been placed on larger levels. If there are more than ω such vertices, the vertices with the largest labels are placed first and the remaining vertices are placed on the next level to proceed. See [29, 32] for details. The algorithm gives a $(2 - \frac{2}{\omega})$-approximation of the optimal height of the leveling [98].

In the cyclic case, Problem 3 is again trivial: Such a cyclic leveling exists simply if $|V| \leq \omega \cdot k$, i. e., if there is enough space for all vertices. It can be constructed by arbitrarily placing vertices within the $\omega \times k$ grid.

3.2.2 Sum of the Edge Spans

Next we present a problem treating the span of the leveling.

Problem 4. *Let $l \in \mathbb{N}$ $(l > 0)$. Does there exist a leveling of G with* $\operatorname{span}(G) \leq l$*?*

Note that minimizing the span is equivalent to minimizing the number of dummy vertices. In the hierarchical case Problem 4 can be solved by an integer linear program (ILP) [66] which minimizes the span of G:

$$\min \sum_{(u,v)\in E} (\phi(v) - \phi(u)) \tag{3.1}$$

$$\forall v \in V : \phi(v) \in \mathbb{N} \setminus \{0\} \tag{3.2}$$

$$\forall e = (u, v) \in E : \phi(v) - \phi(u) \geq 1 \tag{3.3}$$

This ILP can be solved in polynomial time since the constraint matrix is totally unimodular [133]. 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, alternatively, the modulo operation is needed. However, if the number of levels is not fixed, minimal span is always achieved by using only one level. All edges have span one then. Unfortunately, such a drawing is difficult to understand as all edges wrap around the center completely. Therefore, we specialize Problem 4 as follows:

Problem 5. *Let $l, k \in \mathbb{N}$ $(l, k > 0)$. Does there exist a leveling of G on k levels and* $\operatorname{span}(G) \leq l$*?*

In the hierarchical case, the ILP solving Problem 4 can still be used as minimizing the span minimizes the number of levels as well [50, 93]. If the minimal number of levels is smaller than k, some levels remain unused, however.

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

Problem 6 (Bipartite subgraph [69])**.** *Let $G = (V, E)$ be an undirected graph and $l \in \mathbb{N}$. Does there exist a bipartite subgraph G' of G with at least l edges?*

Lemma 3.1. *Let $G = (V, E)$ be an undirected graph and $l \in \mathbb{N}$. Let $G^d = (V, E^d)$ 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^d on two levels with* $\text{span}(G^d) \leq 2|E| - l$.

Proof. "⇒": 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 vertex set with all edges of E' between V_1 and V_2. We construct the following leveling for G^d: For each vertex $v \in V_1$, we set $\phi(v) = 1$, for each vertex $v \in V_2$, we set $\phi(v) = 2$, and for all remaining vertices $v \in V \backslash (V_1 \cup V_2)$, we set $\phi(v) = 1$. Then each edge in E' has span one and all other edges have span one or two. Hence, $\text{span}(G^d) \leq |E'| + 2(|E| - |E'|) = 2|E| - |E'| \leq 2|E| - l$.

"⇐": Let ϕ be a leveling of G^d with $\text{span}(G^d) \leq 2|E| - l$. Let V_1 and V_2 be the vertices of V on level 1 and 2, respectively. Let $E' \subseteq E$ be the set of edges e such that one end vertex is in V_1 and the other is in V_2. Then, $G' = (V, E')$ is bipartite. All edges in E' have span one in the leveling ϕ and all other edges have span 2. As $\text{span}(G^d) \leq 2|E| - l = 1 \cdot l + 2(|E| - l)$, there are at least l edges with span one. 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 any fixed $k > 2$:

Problem 7 (Graph k-colorability [69])**.** *Let $G = (V, E)$ be an undirected graph and let $k \in \mathbb{N}$. Does there exist a coloring $c : V \to \{1, \ldots, k\}$, such that $c(u) \neq c(v)$ for every edge $e = \{u, v\} \in E$?*

Lemma 3.2. *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, $\text{span}(G') \geq k \cdot \frac{|E'|}{2} = k \cdot |E|$.

"⇒": Let c be a coloring of G. Set $\phi = c$. Then, for each edge with end vertices 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.

"⇐": 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. Consequently, $c = \phi$ is a correct coloring. □

Theorem 3.1. *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 even for a fixed k.*

Proof. Lemma 3.1 and Lemma 3.2 show that the problem is $\mathcal{NP}$-hard for $k = 2$ and $k > 2$, respectively. The problem is obviously in $\mathcal{NP}$. □

3.2.2.1 Restricted Width

Next, we additionally restrict the width of the leveling.

Problem 8. *Let $l, \omega, k \in \mathbb{N}$ $(l, \omega, k > 0)$. Does there exist a leveling of G on k levels with width at most ω and* $\text{span}(G) \leq l$*?*

In the hierarchical case, the problem is the directed optimal linear arrangement problem for $\omega = 1$ which is $\mathcal{NP}$-hard [54, 69]. Hence, Problem 8 is $\mathcal{NP}$-hard for arbitrary width as well.

Problem 9 (Directed optimal linear arrangement [54, 69])**.** *Let $G = (V, E)$ be a directed acyclic graph and $l \in \mathbb{N}$. Does there exist a bijective function $f : V \to \{1, \ldots, |V|\}$ with $f(u) < f(v)$ for each edge $(u, v) \in E$ and $\sum_{(u,v)\in E} f(v) - f(u) \leq l$?*

In the cyclic case Problem 8 is the $\mathcal{NP}$-hard minimum circular arrangement problem [65, 100] for $\omega = 1$ and, thus, $\mathcal{NP}$-hard for arbitrary width.

Problem 10 (Minimum Circular Arrangement [65, 100])**.** *Let $G = (V, E)$ be a directed graph and $l \in \mathbb{N}$. Does there exist a bijective function $f : V \to \{1, \ldots, |V|\}$ such that $\sum_{(u,v)\in E} \text{span}(f(u), f(v)) \leq l$ where $\text{span}(a, b) = b - a$ if $b > a$ and $\text{span}(a, b) = b - a + |V|$ if $b \leq a$?*

3.2.3 Maximal Edge Span

Problem 11. *Let $l \in \mathbb{N}$ $(l > 0)$. Does there exist a leveling of G with* $\text{span}(e) \leq l$ *for each edge $e \in E$?*

The problem is $\mathcal{NP}$-hard in the hierarchical setting if we allow only one vertex per level, as it is the $\mathcal{NP}$-hard directed bandwidth problem then [68, 69].

Problem 12 (Directed bandwidth [68, 69])**.** *Let $G = (V, E)$ be a directed graph and $l \in \mathbb{N}$ $(0 < l < |V|)$. Does there exist a bijective function $f : V \to \{1, \ldots, |V|\}$ with $f(u) < f(v)$ and $f(v) - f(u) \leq l$ for each edge $e = (u, v) \in E$?*

We now show that Problem 11 is $\mathcal{NP}$-hard for each fixed width ω.

Problem 13 (Hierarchical max span leveling with width constraint)**.** *Let $G = (V, E)$ be a directed acyclic graph and $l \in \mathbb{N}$ and $\omega \in \mathbb{N}$ $(0 < l < |V|, \omega > 0)$. Does there exist a hierarchical leveling ϕ with width at most ω such that each edge has a span of at most l?*

Theorem 3.2. *Hierarchical max span leveling with width constraint is $\mathcal{NP}$-complete even for a fixed width ω.*

Proof. The problem is obviously in $\mathcal{NP}$. To prove that it is $\mathcal{NP}$-hard, we reduce from the directed bandwidth problem. Let $G = (V, E)$ be a directed acyclic graph and $l \in \mathbb{N}$, $0 < l < |V|$. First, we assume G to be connected. The maximum number of levels G can use in a leveling is $m = (|V| - 1)l + 1$. We construct G' by adding $\omega - 1$ copies of a graph K. K consists of vertices $v_1, \dots v_{2m-1}$ and edges (v_i, v_j) for each $1 \leq i < j \leq 2m - 1$ with $j - i \leq l$. Intuitively, each vertex v_i has an edge to the next l vertices if they exist. The only way to level each K with span at most l for each edge with arbitrary width is to use $2m - 1$ consecutive levels and to put the vertices on these levels in order of their indices. Let u be an arbitrary vertex of G. For each K, we add the edges (v_{m-l}, u) and (u, v_{m+l}). Hence, vertex u has to be leveled next to the vertex v_m of each K. Then, $m - 1$ levels above u and below u are occupied by the $\omega - 1$ copies of K and only one position per level remains.

The bandwidth function f of G and the leveling of the subgraph G of G' can then easily be transformed in both directions. Note that the leveling of G can contain empty levels. However, removing them only decreases the span of some edges. Consequently, G has directed bandwidth at most l if and only if G' has a leveling of width ω such that each edge has a span of at most l.

If G is not connected, a very similar construction can be applied by placing each connected component next to even larger graphs K far enough apart from each other. □

In the cyclic case, using only one level gives the optimal result with all edges having span one again. Therefore, we specialize Problem 11 to include the number of levels k to use:

Problem 14. *Let $l, k \in \mathbb{N}$ $(0 < l < k)$. Does there exist a leveling of G with exactly k levels and $\operatorname{span}(e) \leq l$ for each edge $e \in E$?*

In the case of width one, bandwidth problems are again very similar to Problem 14. Cyclic bandwidth problems have been discussed [97, 101] in the undirected case. Here, we define a directed version:

Problem 15 (Directed cyclic bandwidth)**.** *Let $G = (V, E)$ be a directed graph and $l \in \mathbb{N}$ $(0 < l < |V|)$. Does there exist a bijective function $f : V \to \{1, \dots, |V|\}$ such that for each edge $e = (u, v) \in E$, $f(v) - f(u) \leq l$ if $f(v) > f(u)$ and $f(v) - f(u) + |V| \leq l$ if $f(v) < f(u)$.*

Using our notation, the problem of directed cyclic bandwidth is to find a cyclic leveling of G on $|V|$ levels with width one such that for each edge e $\operatorname{span}(e) \leq l$.

Theorem 3.3. *Directed cyclic bandwidth is $\mathcal{NP}$-complete.*

Proof. The problem is obviously in $\mathcal{NP}$. To prove that it is $\mathcal{NP}$-hard we reduce from the directed bandwidth problem. Let $G = (V, E)$ be a directed graph and

$l \in \mathbb{N}$, $0 < l < |V|$. We construct $G' = (V', E')$ by adding a new graph K consisting of l vertices $v_1, \dots v_l$ and edges (v_i, v_j) for each $1 \le i < j \le l$. We show that G has directed bandwidth $\le l$ if and only if G' has directed cyclic bandwidth $\le l$. Note that the only possible assignment of levels to the vertices in K is to use l consecutive levels. Hence, no edge of G can start before K and end after K.

"$\Rightarrow$": Let f be a hierarchical leveling of G. We define $f'(v) = f(v)$ for all $v \in V$ and $f'(v_i) = |V| + i$ for each v_i from K. Then, the span of each edge $e \in E$ is not changed and, hence, at most l. For each edge e from K, span$(e) \le l$ holds as well. Hence, f' is a correct cyclic leveling with each edge $e \in E'$ having span at most l.

"$\Leftarrow$": Let f' be a cyclic leveling of G'. W. l. o. g., let $|V|+1, \dots, |V'|$ be the levels of the vertices of K. Then $f(v) = f'(v)$ is a correct leveling of G with each edge $e \in E$ having span at most l. □

Next, we specialize Problem 15 using arbitrary but fixed width ω.

Problem 16 (Cyclic max span leveling with width constraint)**.** *Let $G = (V, E)$ be a directed graph and $l, \omega, k \in \mathbb{N}$, $(0 < l < |V|, \omega, k > 0)$. Does there exist a leveling $\phi : V \to \{1, \dots, k\}$ with width at most ω such that each edge has a span of at most l?*

Lemma 3.3. *Let $k, l \in \mathbb{N}$ $(0 < l < k)$. Then there exists a graph $K_{k,l}$ with k vertices that has a cyclic leveling on k levels with span at most l for each edge such that in all such cyclic levelings of $K_{k,l}$ exactly one vertex is placed on each level.*

Proof. $K_{k,l}$ consists of the vertices $v_1, \dots, v_k$ and the edges $(v_i, v_{((i+j-1) \bmod k)+1})$ for each $1 \le i \le k$ and $1 \le j \le l$. Intuitively, when interpreting the vertices v_i ordered in a cyclic manner, each vertex has an edge to the next l vertices. Obviously, $K_{k,l}$ can be leveled with width one and span at most l for each edge by placing each vertex v_i on level i. We now show that it is not possible to place two vertices of $K_{k,l}$ on the same level.

First, let $l = k-1$. Then the underlying undirected graph of $K_{k,l}$ is complete. If two vertices are placed on the same level, the edge between them would have span $k > l$. Hence, on each level only one vertex can be placed.

Let $l < k-1$ and ϕ be a leveling of $K_{k,l}$. W. l. o. g., let $\phi(v_1) = 1$. We prove that $\phi(v_2) = 2$. The vertex v_1 has l outgoing edges ending at the vertices $v_2, \dots, v_{l+1}$. These vertices have to be on the levels 2 through $l+1$ as each edge has span l at most. Note that these vertices are pairwise connected. Hence, no two vertices of $v_2, \dots, v_{l+1}$ can be placed on the same level as the edge between them would have span $k > l$ then. Consequently, exactly one of the vertices $v_2, \dots, v_{l+1}$ is on each of the levels 2 through $l+1$. Assume for contradiction that $\phi(v_2) = m \neq 2$. Then there exists one vertex $v_j \in \{v_3, \dots, v_{l+1}\}$ with $\phi(v_j) = m-1$. However, the edge (v_2, v_j) has span $k-1 > l$ then, which is a contradiction. By induction, $\phi(v_3) = 3, \dots, \phi(v_k) = k$ follows. Hence, on each level only one vertex can be placed. □

Theorem 3.4. *Cyclic max span leveling with width constraint is $\mathcal{NP}$-complete even for a fixed width ω.*

Proof. The problem is obviously in $\mathcal{NP}$. To prove that it is $\mathcal{NP}$-hard we reduce from the directed cyclic bandwidth problem. Let $G = (V, E)$ be a directed graph and $l \in \mathbb{N}$, $0 < l < |V|$. We construct G' by adding $\omega - 1$ copies of the graph $K_{|V|,l}$. These $\omega - 1$ copies occupy $\omega - 1$ positions on each level due to Lemma 3.3. Hence, for the original graph G exactly one position on each level remains. Thus, G has directed cyclic bandwidth at most l if and only if G' has a leveling on $|V|$ levels with width ω such that each edge has a span of at most l. □

Note that the practical use of this problem is limited: Each cycle in a graph has to wrap around the center at least once. Hence, if a graph containing a cycle using c edges is leveled on k levels, the maximal span of the edges is at least $\lceil \frac{k}{c} \rceil$. All other edges belonging to longer cycles are not optimized regarding their length then. One approach is to minimize the maximal edge length first and then try to minimize the sum while respecting the maximal edge length. Nevertheless, we present three distinct heuristics in the next section which try to minimize the sum of all edge spans while staying within a given width ω.

3.3 Heuristics for Cyclic Leveling

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 [93] or the Coffman-Graham algorithm [29] 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 vertices with no incoming or outgoing edges. As it is not guaranteed that such vertices exist in the cyclic case at all, new approaches have to be developed. We introduce three novel heuristics with an experimental evaluation in Section 3.4.

The input to the algorithms are the directed graph $G = (V, E)$, the number of levels k, and the maximum number of vertices on a level ω. The output is the leveling $\phi : V \to \{1, \ldots, k\}$.

3.3.1 Breadth First Search

The *breadth first search heuristic* (BFS) (see Algorithm 3.1) is rather simple: We choose an arbitrary start vertex v, set $\phi(v) = 1$, and perform a directed BFS from v. When we reach a vertex w for the first time using an edge (u, w), we set $\phi(w) = \text{next}(\phi(u))$ (line 16) if this level does not contain already ω vertices. Otherwise, we move w to the next non-full level. If not all vertices are reached from the first start vertex, we start again with an unleveled vertex (line 6) and place it on the first non-full level. A variant is to use a random non-full level here.

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

Algorithm 3.1. breadthFirstSearchLeveling

```
Input: $G$: a directed graph, $k$: the number of levels,
       $\omega$: the maximum number of vertices 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         $updateN(\phi(u))$
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 $\phi$
```

Lemma 3.4. *The BFS leveling heuristic has a time complexity of $\mathcal{O}(|V|+|E|+k^2)$.*

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. □

To update the array N at position i (see line 10 in Algorithm 3.1), we traverse the levels starting at i downwards until we find the first non-full level j. We then traverse the levels from i upwards and update for each level l with $N[l] = i$ this value to $N[l] = j$. We stop when we reach the first non-full level. In many cases this update is done in constant time. Therefore, more sophisticated solutions like pointer doubling do not seem to be necessary. Furthermore, if the number of levels k is in $\mathcal{O}(\sqrt{|V|})$, the term k^2 does not increase the time complexity at all. This is reasonable as choosing k and the width of the leveling ω in that magnitude results in a good aspect ratio of the leveling.

3.3.2 Minimum Spanning Tree

This heuristic has similarities to the algorithm of Prim [30, 120], 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. When we level a vertex v, the span of all edges in $E(v, V')$ is determined. Therefore, we set $\phi(v)$ such that $\text{span}(E(v, V'))$ is minimized. Note that there are possibly more edges incident to v which are also incident to not yet leveled vertices. Such an edge will be considered when its other end vertex is leveled.

Beside the question where to put the next vertex, we also have to determine which vertex to level next. To decide this, we use a distance function $\delta(v)$ which is defined for each not yet leveled vertex v. We set $\delta(v) = \infty$ for all vertices with $\deg(v) = 0$ and discuss four options for δ:

3.3.2.1 Minimum Increase in Span (MST_MIN)

We choose the vertex which creates the minimum increase in span in the already leveled graph:

$$\delta_{\text{MIN}}(v) = \min_{\phi(v) \in \{1,\ldots,k\}} \text{span}(E(v, V')) \tag{3.4}$$

3.3.2.2 Minimum Average Increase in Span (MST_MIN_AVG)

By using the distance function δ_{MIN}, vertices with a low degree are placed first as vertices with a high degree will almost always cause a high increase in span. Therefore, considering the increase in span per edge is reasonable:

$$\delta_{\text{MIN\_AVG}}(v) = \min_{\phi(v) \in \{1,\ldots,k\}} \frac{\text{span}(E(v, V'))}{|E(v, V')|} \tag{3.5}$$

3.3.2.3 Maximum Increase in Span (MST_MAX)

Choose the vertex which causes the maximum increase in span per edge:

$$\delta_{\text{MAX}}(v) = \frac{1}{\delta_{\text{MIN}}(v)} \tag{3.6}$$

The idea behind this is as follows: A vertex which causes a high increase in span will cause this increase when leveled later as well. But if we level this vertex earlier, we can possibly level other adjacent, not yet leveled vertices in a better way.

3.3.2.4 Maximum Average Increase in Span (MST_MAX_AVG)

Choose the vertex which causes the maximum average increase in span per edge:

$$\delta_{\text{MAX\_AVG}}(v) = \frac{1}{\delta_{\text{MIN\_AVG}}(v)} \tag{3.7}$$

Algorithm 3.2. minimumSpanningTreeLeveling

Input: G: a directed graph, k: the number of levels,
ω: the maximum number of vertices on each level

Output: ϕ: 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 $\phi$
```

This distance function combines the ideas of considering the average increase (like $\delta_{\text{MIN\_AVG}}$) and choosing vertices with a high increase first (like δ_{MAX}). Note that we only use the distance function $\delta(v)$ to determine which vertex to level next. When we level a vertex v, we set $\phi(v)$ such that the increase in span will be minimized.

In some cases, several levels for v create the same increase in span. We then choose a level for v which minimizes $\sum_{e \in E(v,V')} \text{span}(e)^2$ as well. Thus, we assign to v a level which is centered between its leveled adjacent vertices. In each case, we can only use a level which is not yet full, i. e., does not have ω vertices on it. For each vertex, which has at least one already leveled neighbor, its optimal level is calculated. To ensure that the vertex can be placed on that level when it is its turn, each such vertex blocks one position on its optimal level. Algorithm 3.2 shows the complete heuristic, where the status *black* indicates a leveled vertex, the status *gray* indicates that a neighbor of the vertex is already leveled, and the status *white* is assigned to all other vertices.

Lemma 3.5. *The MST heuristic needs $\mathcal{O}(|V|\log|V| + k \cdot \deg(G) \cdot |E|)$ time.*

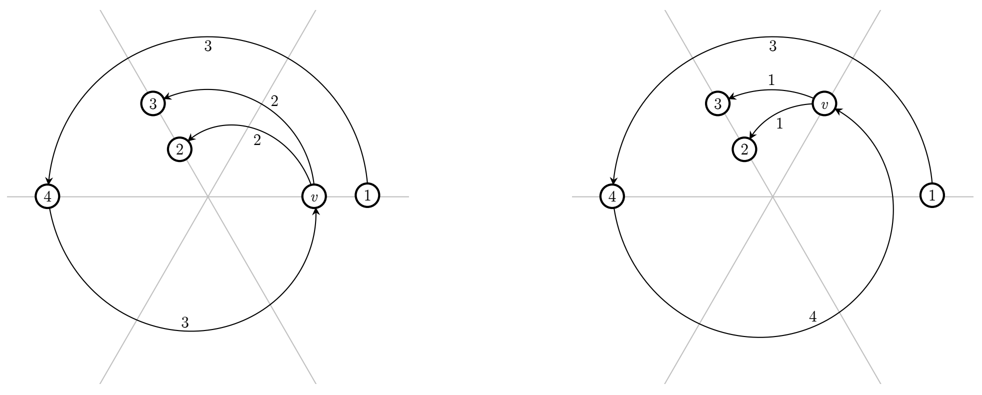
Proof. The time complexity is dominated by the while loop. Here, removing all vertices from the heap costs $\mathcal{O}(|V|\log|V|)$ in total. Each edge $e = (w, z) \in E$ may change its span whenever a neighbor v of w (or z) is assigned to a level. In this case, each of the k levels is considered as a possible level for w (or z). Hence, 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 [62]). □

Note that the time complexity of this approach is higher than the complexity of computing a minimal spanning tree as our heuristic borrows the basic concept only.

3.3.3 Force-Based

Spring embedder (SE) physically model the edges as springs which connect the vertices [40, 64, 93]. 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:

$$\text{force}(v) = \sum_{(v,w)\in E} (\text{span}(v,w) - 1)^2 - \sum_{(u,v)\in E} (\text{span}(u,v) - 1)^2 \qquad (3.8)$$



(a) v in energy minimum of (3.8) (b) v in energy minimum of (3.9)

Figure 3.3. Force-based placement of vertex v

However, moving a vertex to its energy minimum using this force does not minimize the span of the graph, i. e., (3.8) minimizes the deviation between the edge lengths, e. g., see Figure 3.3. 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 these problems by directly using the span as the (undirected) force which has to be minimized:

$$\text{force}(v) = \text{span}(E(v, V)) \tag{3.9}$$

We move the vertex which has the maximum force, and we directly move the vertex to its energy minimum, which is the level that minimizes the span. For this, we test all possible, i. e., non-full levels. Note that moving all vertices at once would not decrease time complexity here. Algorithm 3.3 shows a detailed description of the heuristic.

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 3.3. forceBasedLeveling

```
Input: $G$: a directed graph, $k$: the number of levels,
       $\omega$: the maximum number of vertices 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 3.6. *In the force-based heuristic $\mathcal{O}(|V| \log |V| + k \cdot |E|)$ time is needed for each iteration.*

Proof. Inserting all vertices into the heap can be implemented in $\mathcal{O}(|V|)$ time. Removing all vertices from the heap has time complexity $\mathcal{O}(|V| \log |V|)$ in total. Computing the energy-minimal level for v costs $\mathcal{O}(k \cdot \deg(v))$, which is $\mathcal{O}(k \cdot |E|)$ for all vertices. 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 [62]). □

3.4 Experimental Results

In this section, we evaluate the heuristics by comparing them to each other and to an optimal leveling. The optimal leveling is computed by a branch and bound algorithm which can be used for graphs up to 18 vertices. We use the span of found levelings as upper bounds and the span of a partial solution plus the number of unleveled edges as its lower bound.

In Figure 3.4, the running time of the algorithms are shown. Figure 3.5 compares the calculated spans of the heuristics with the optimal span. For a better pairwise comparison of the heuristics, Figure 3.6 only shows their results on larger graphs.

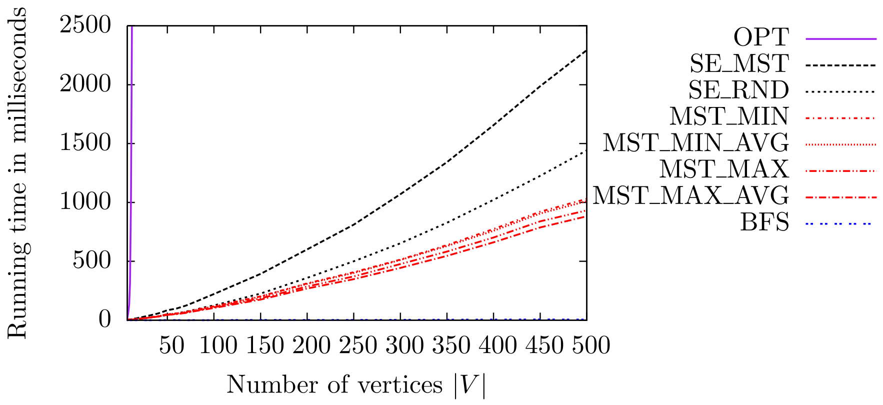


Figure 3.4. Running times of the algorithms

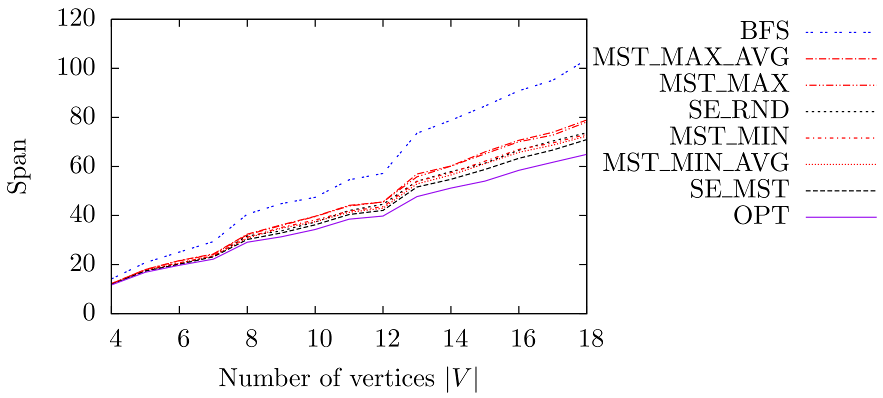


Figure 3.5. Average spans of small graphs

For Figure 3.4 and 3.6, the number of vertices $|V|$ was increased by steps of 50 each time. For each size ten graphs with $|E| = 5|V|$ were created randomly. For Figure 3.5, ten graphs for each size $|V|$ and $|E| = 2|V|$ were used. In all three diagrams, k and ω were set to $\lceil \sqrt{2|V|} \rceil$, such that there were $2|V|$ possible vertex positions. Each heuristic was applied to each graph $|V|$ times using different start vertices or initial levelings. For graphs smaller that 30 vertices each heuristic was

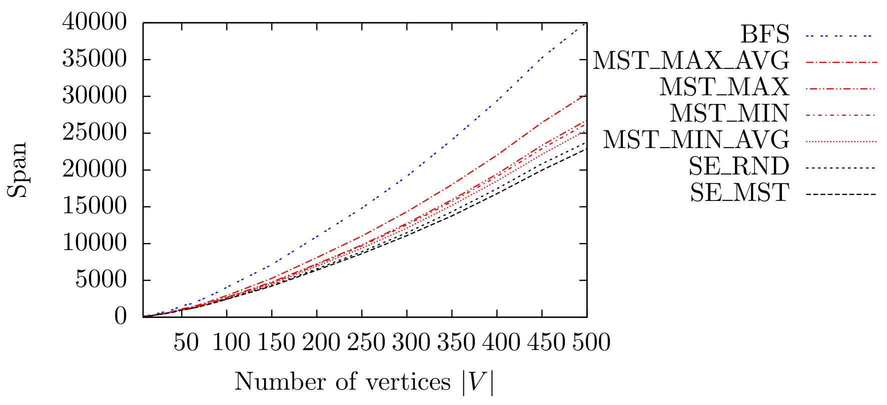


Figure 3.6. Average spans of large graphs

applied 30 times. The diagrams show the average results of these iterations. 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 [7].

The benchmarks show the practical performance of the algorithms. As expected, the simple breadth first search heuristic gives the poorest results but has the smallest running time. The MST and force-based heuristics perform considerably better, where the SE algorithms give better results than the MST variants. All MST versions do not differ very much, but MST_MIN_AVG seems to be the best. Unsurprisingly, combining the force-based heuristic with MST initialization computes the best leveling. The result is even close to the optimum.

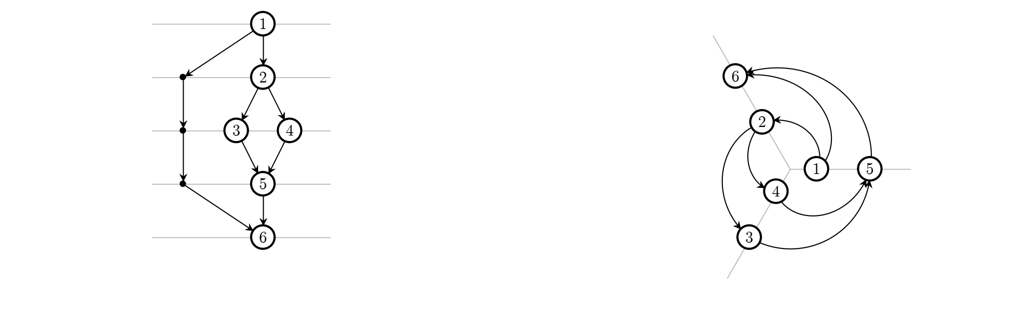
In the resulting levelings, many levels contain dummy vertices only. Hence, $\lceil\sqrt{2|V|}\rceil$ levels seem to be too much when a width of $\lceil\sqrt{2|V|}\rceil$ is given. In this chapter we compare cyclic leveling algorithms only. Hence, all heuristics had to deal with this number of levels. However, we use $\lceil\sqrt{0.8|V|}\rceil$ as the number of levels in Chapter 7 to have a fair comparison of the cyclic and hierarchical framework.

The results can be improved by applying the heuristics i times to the same graph with i different start vertices or different initial levelings and choosing the best result. However, the price is an i times higher running time.

3.5 Incorporating the Width of Dummy Vertices

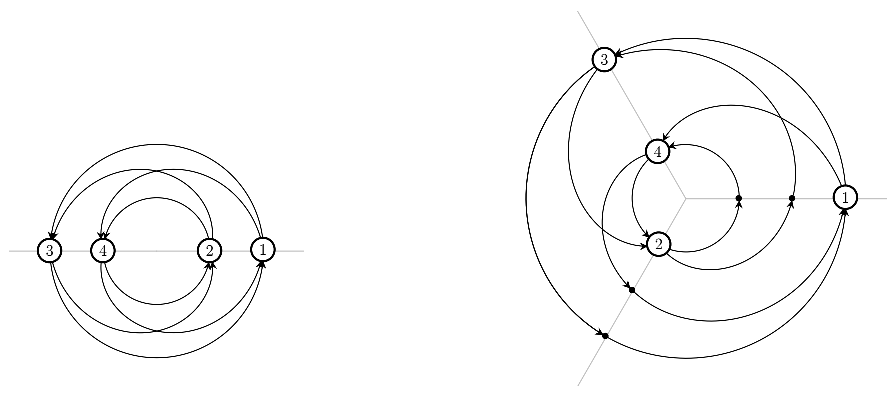
In most cases, dummy vertices are ignored when computing the width of a leveling. For many applications this is reasonable as the original vertices are labeled and, therefore, need much more space than the dummy vertices which are replaced by bends in the final drawing anyway. However, even for these bends a minimum distance is needed. First, we state several differences between the hierarchical and cyclic leveling when treating the width of dummy vertices.

Examine the acyclic graph in Figure 3.7. When leveling the graph hierarchically, the minimum width is three. See Figure 3.7(a) for a possible leveling. When leveling



(a) Hierarchical leveling with width three (b) Cyclic leveling with width two

Figure 3.7. Levelings of an acyclic graph



(a) Two levels and width two (b) Three levels and width three

Figure 3.8. Levelings of a cyclic graph with different numbers of levels

the graph in a cyclic way, width two is sufficient (see Figure 3.7(b)). Hence, acyclic graphs can have a smaller width in cyclic levelings than in the hierarchical case.

The number of levels is more important in the cyclic setting than in the hierarchical one. Survey the graph in Figure 3.8. It can be leveled on two levels with width two (see Figure 3.8(a)). However, if there are three levels, width two does not suffice. Note that the graph contains four edge disjoint cycles with two vertices each. Each of them has to wrap around the center at least once and has at least one dummy vertex. Hence, there are at least eight vertices and a width of three is needed. Figure 3.8(b) shows such a leveling. In contrast, having too many levels is never a problem in the hierarchical case (if empty levels are allowed).

Another problem are non-connected graphs. In the hierarchical case, a graph has a leveling with width at most ω if and only if each of its connected components has a leveling with width at most ω. The graphs can then be placed one below the other. In the cyclic leveling problem, this is not the case. Each cyclic connected component needs some space on each level. Therefore, the connected components cannot be treated independently.

We now state the following problem:

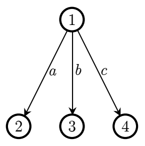
Problem 17. *Let $\omega \in \mathbb{N}, \omega_v, \omega_d \in \mathbb{R}$ $(\omega \geq \omega_v, \omega_d > 0)$. Does there exist a leveling of a directed graph G with width at most ω if each original vertex $v \in V$ has width ω_v and each dummy vertex has width ω_d?*

In the hierarchical case, Problem 17 is $\mathcal{NP}$-hard [20]. However, it is fixed parameter tractable [37], which is essentially a result of [20] that we state here more clearly:

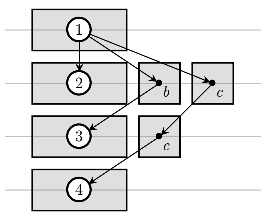
Theorem 3.5. *Let $\omega \in \mathbb{N}, \omega_v, \omega_d \in \mathbb{R}$ $(\omega \geq \omega_v, \omega_d > 0)$. Let $a = \frac{\omega}{\min(\omega_v, \omega_d)}$ be a constant. Let $G = (V, E)$ be a directed acyclic graph. Whether there is a hierarchical leveling of G with width at most ω, if each original vertex $v \in V$ has width ω_v and each dummy vertex has width ω_d, can be decided in polynomial time.*

Proof. We treat each connected component C of G separately. The constant a depicts how many (dummy) vertices can be placed on the same level at most. We build a new graph $G' = (V', E')$. Each vertex $l' \in V'$ defines a level of a potential leveling of C. Formally, each vertex $l' \in V'$ is a subset of $V \cup E$ with at most a members. The vertices $v \in V$, which are in l', are the vertices on the level represented by l'. The edges $e \in E$ in l' are the edges that cross the level and, hence, have a dummy vertex on it. Note that we can restrict V' to the levels containing at least one original vertex. The overall number of potential levels $|V'|$ is then at most $(|V| + |E|)^a$. It is easy to check whether a level l'_2 can be a direct successor of a level $l'_1 \neq l'_2$: The sets of contained original vertices have to be disjoint, for each original vertex u in l'_1 with an outgoing edge (u, w), l'_2 has to contain either the vertex w or the edge (u, w) and vice versa, and for each edge $e = (u, w)$ in l'_1, l'_2 has to contain either the vertex w or the edge (u, w) and vice versa. If these conditions hold, an edge (l'_1, l'_2) is added to E'. The connected component has a leveling with width at most ω if and only if there exists a path in G' from the empty level to the empty level, i. e., a cycle containing the empty level, using at least one edge. Note that such a path contains all original vertices at least once as C is connected. An arbitrary path can contain C several times. This happens only if all these copies of C can be placed next to each other within the width ω. Hence, a leveling of C with width ω exists in this case as well. Using a shortest path avoids this problem and gives a leveling with minimal height for the connected component C. The vertices of the path represent the individual levels of the leveling. □

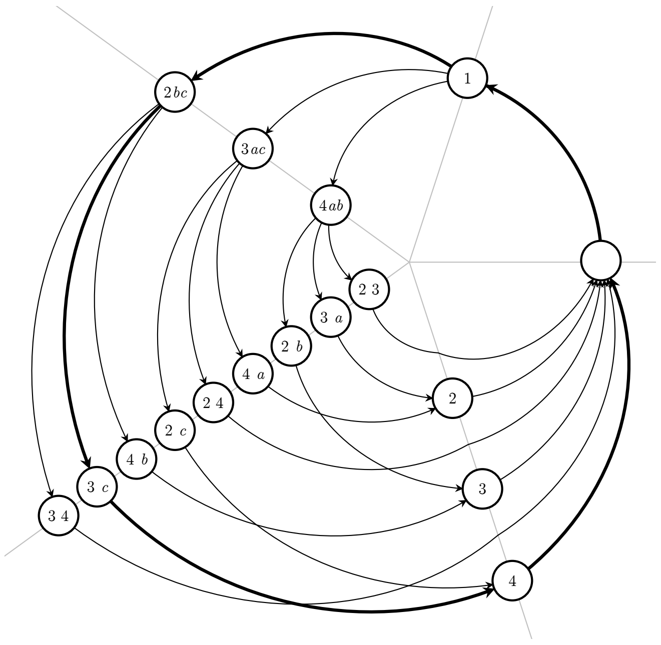
Computing the shortest path gives a leveling of width ω with minimal height for one connected component. However, this does not automatically yield the optimal height for the complete graph. If a maximum height is given as well, the algorithm can be used if the graph is connected only [20]. See Figure 3.9 for an example. The input graph G in Figure 3.9(a) is to be leveled within width $\omega = 4$. Each original vertex has the width $\omega_v = 2$ and each dummy vertex the width $\omega_d = 1$. Figure 3.9(c) shows the graph G' which represents all possible levelings of G within these restrictions. Here, vertices representing levels unreachable from the empty level like the level $\{1, 2\}$ are omitted. The empty level is part of each cycle in G'. Hence, any cycle of G' can be chosen as a leveling of G. The leveling belonging to the cycle $\emptyset, \{1\}, \{2, b, c\}, \{3, c\}, \{4\}, \emptyset$ is presented in Figure 3.9(b). A shortest cycle like $\emptyset, \{1\}, \{2, b, c\}, \{3, 4\}, \emptyset$ represents a leveling with the minimal number of levels.



(a) Input graph G



(b) Resulting leveling



(c) Graph G' representing all possible levelings of G

Figure 3.9. Example of finding a leveling with width of dummy vertices

In the cyclic case, Problem 17 is $\mathcal{NP}$-hard as well. We reduce from the hierarchical leveling problem with dummy vertices with width. If an acyclic graph G can be leveled in the hierarchical case with width ω it can be leveled in the cyclic case with width ω as well if there are enough levels. However, this does not hold the other way round as can be seen in Figure 3.7. The idea for the reduction is to add another graph T which *blocks* at least one level completely, i. e., no additional vertex or dummy vertex can be placed on the level. Intuitively, the graph T removes the possibility for G to be leveled in a cyclic way. We first show that such a graph always exists.

Lemma 3.7. *Let $\omega \in \mathbb{N}, \omega_v, \omega_d \in \mathbb{R}$ $(\omega \geq \omega_v, \omega_d > 0)$. There exists a tree $T = (V, E)$ that can be leveled (hierarchically or cyclically) respecting the width ω if each original vertex $v \in V$ has width ω_v and each dummy vertex has width ω_d such that in any leveling of T at least one level is blocked.*

Proof. We differentiate between $\omega_v < \omega_d$ and $\omega_v \geq \omega_d$ and consider $\omega_v < \omega_d$ first. Let T be a tree with root r and $\lfloor \frac{\omega}{\omega_v} \rfloor$ children. Let r be leveled on an arbitrary level. One possibility is to level all children on the next level j using width $\lfloor \frac{w}{\omega_v} \rfloor \omega_v > \omega - \omega_v > \omega - \omega_d$. Then this level is blocked, as no other original vertex or dummy vertex can be placed on this level. If not all children are leveled on level j then this results in replacing some original vertices by dummy vertices on level j. As $\omega_d > \omega_v$ the level j is blocked then as well.

Let $\omega_v \geq \omega_d$. Let T be a tree with root r and $\lfloor \frac{\omega - \omega_v}{\omega_d} \rfloor + 1$ children. Let r be leveled on an arbitrary level i. Let j be the first level below i containing a child of r. On j for each child either the child itself or a dummy vertex of its incoming edge has to be placed. If j contains exactly one child all these vertices occupy the width $\omega_v + \lfloor \frac{\omega - \omega_v}{\omega_d} \rfloor \omega_d > \omega - \omega_d > \omega - \omega_v$. Hence, no further dummy vertex or original vertex can be placed on this level. If more original vertices are placed on level j the width increases even more. It remains to be shown that T can be leveled at all respecting the width constraints. This can be done, e. g., by placing one child on each subsequent level. □

For an example, let $\omega = 4$, $\omega_v = 2$, and $\omega_d = 1$. As $\omega_v \geq \omega_d$, the tree T has $\lfloor \frac{\omega - \omega_v}{\omega_d} \rfloor + 1 = 3$ children as in Figure 3.9(a). A leveling within the given constraints which blocks the level of the first child is presented in Figure 3.9(b).

Theorem 3.6. *Let $\omega \in \mathbb{N}, \omega_v, \omega_d \in \mathbb{R}$ $(\omega \geq \omega_v, \omega_d > 0)$. Deciding whether there exists a cyclic leveling of a directed graph G with width at most ω if each original vertex $v \in V$ has width ω_v and each dummy vertex has width ω_d is $\mathcal{NP}$-complete.*

Proof. Let G^h be an acyclic graph. Let G be the graph consisting of G^h and the tree T of Lemma 3.7. We prove that G^h has a hierarchical leveling following the given width constraints if and only if G has a cyclic leveling following the constraints.

"$\Rightarrow$": Let G^h have a hierarchical leveling on l_{G^h} levels respecting the given width constraints. Due to Lemma 3.7, T can be leveled hierarchically with these width constraints as well. Let l_T be the number of levels needed for T. Then G can be leveled on $l_{G^h} + l_T$ levels in a cyclic way within the width constraints by, e. g., placing T below G^h.

"$\Leftarrow$": Let G have a cyclic leveling on l_G levels respecting the given width constraints. Due to Lemma 3.7, T blocks at least one level completely. W. l. o. g., let l_G be this level. Hence, G^h only uses the levels 1 through $l_G - 1$ at most, and the leveling of G^h is already a hierarchical leveling.

Using more than $|V|$ levels cannot reduce the width of a leveling. Hence, a potential leveling can be created (and verified) in polynomial time and the problem is in $\mathcal{NP}$. □

The problem remains fixed parameter tractable for connected graphs.

Theorem 3.7. *Let $\omega \in \mathbb{N}, \omega_v, \omega_d \in \mathbb{R}$ $(\omega \geq \omega_v, \omega_d > 0)$. Let $a = \frac{\omega}{\min(\omega_v, \omega_d)}$ be a constant. Let $G = (V, E)$ be a connected directed graph. Whether there is a cyclic leveling of G with width at most ω if each original vertex $v \in V$ has width ω_v and each dummy vertex has width ω_d can be decided in polynomial time.*

Proof. We construct the same graph G' as in the proof of Theorem 3.5. G has a cyclic leveling with width at most ω if and only if there exists a cycle in G' using at least one edge. □

Note that this time, the empty level does not have to be a part of the cycle. Finding the shortest cycle gives a leveling with width at most ω and the minimal number of levels.

3.6 Summary

We first examined the complexity of several leveling problems in the cyclic case. Restricting the width or height of a leveling is trivial in the cyclic case. In contrast to the hierarchical case, minimizing the span of the graph is $\mathcal{NP}$-hard for a fixed number of levels $k > 1$. Minimizing the maximum edge span is $\mathcal{NP}$-hard for a fixed width ω. Deciding whether there is a leveling of width ω, if dummy vertices are considered as well, is $\mathcal{NP}$-hard in general. However, it is solvable in polynomial time if the width ω is fixed. Tab. 3.1 gives a more detailed overview of the results.

Common leveling heuristics for the hierarchical Sugiyama framework [29] rely heavily on the acyclic structure of the input graph. Therefore, completely new approaches had to be used for the cyclic variant. We proposed three cyclic leveling heuristics. The first one is based on breadth first search, which is simple but gives poor results. The second has similarities to the minimal spanning tree algorithm of Prim [30, 120] and the third takes ideas from spring embedder techniques [40, 64]. Both perform considerably better than the BFS approach. The best results were achieved by initializing the spring embedder method with a leveling produced by the MST approach.

3.7 Open Problems

In the hierarchical case several linear time leveling heuristics exist. Finding a practical linear time leveling algorithm in the cyclic case is still an open problem, however.

All presented leveling heuristics require the number of levels k as an additional input. If the number of levels is given by the application (a weekly schedule would use 7 levels) this is no problem but choosing k without context is a non-trivial task. Using a small number of levels leads to short edges. The extreme case is to use only one level, which is always possible. Then each edge has span one, but the drawing will be difficult to understand visually and has more crossings usually. Using a too large number of levels increases the span of the edges and adds many unnecessary dummy vertices.

Obviously, there is some correlation between the number of levels, the width of the levels, and the number of vertices in the graph. The number of levels k times the width ω has to be at least the number of vertices. Normally, choosing $\omega \cdot k$ up to two times as big as needed is sufficient. Testing different values for k and ω and

Table 3.1. Complexity of leveling (with 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 $\phi =$ topsort	Choose injective ϕ
Leveling with k and ω given	$\mathcal{NP}$-hard, [148]	Test $k \cdot \omega \geq \lvert V\rvert$
Minimizing span(G)	$\mathcal{P}$, [66]	Set $k = 1$
Minimizing span(G) with $k > 1$ fixed	$\mathcal{P}$, [66]	$\mathcal{NP}$-hard, Theorem 3.1
Minimizing span(G) with $\omega = 1$	$\mathcal{NP}$-hard, [54]	$\mathcal{NP}$-hard, [65]
Minimizing $\max_{e \in E}$ span(e) with $\omega = 1$, k given	$\mathcal{NP}$-hard, [68]	$\mathcal{NP}$-hard, Theorem 3.3
Minimizing $\max_{e \in E}$ span(e) with $\omega > 1$ fixed, k given	$\mathcal{NP}$-hard, Theorem 3.2	$\mathcal{NP}$-hard, Theorem 3.4
Leveling with ω given, dummy vertices with width	$\mathcal{NP}$-hard, [20]	$\mathcal{NP}$-hard, Theorem 3.6
Leveling with ω fixed, dummy vertices with width	$\mathcal{P}$, [20], Theorem 3.5	$\mathcal{P}$, Theorem 3.7 (connected graphs)

picking the drawing which fits the application best is obviously always a possibility. Another possibility is to use a large k first. Our heuristics tend to leave levels unused, i. e., place only dummy vertices on them, if the number of levels is to high. Hence, removing all levels with only dummy vertices on them could lead to good results. A variant is to level again using the remaining number of levels.

Another approach is to determine the average length of cycles in the input graph. There are many results regarding the length and number of cycles in directed graphs or circles in undirected graphs [1, 13, 73, 84, 145] but finding the average length of cycles seems to be a hard problem. However, in most cases heuristics would suffice.

Another approach to find the number of levels automatically is to try to arrange the vertices without levels first. Edges are drawn counter-clockwise and the span to minimize is the angle the start and end vertex have with the center point. Here, force-based approaches could be used where only incident vertices repel each other. After such a layout is computed vertices in the same sector are placed on the same level. The sectors are made as large as possible without having two incident vertices in the same sector. A study to test these approaches and to investigate other measures which may help finding the number of levels k automatically is needed.

A further open problem is that dummy vertices are not taken into account when computing the width of a level. But considering them is desirable as they clearly add to the width of the final drawing. This problem is a general one as the standard leveling algorithm for the hierarchical Sugiyama framework by Coffman and Graham [29] ignores dummy vertices as well. Recently, two heuristics for hierarchical leveling were proposed [114] which treat dummy vertices. However, the problem remains open for the cyclic case.

The Spring Embedder heuristic could perform even better, if temporarily more vertices per level are allowed than determined by the given width. A existing leveling could unfold then and leave a local minimum more easily.

There are several algorithmic problems similar to the cyclic leveling problem. The circular arrangement problem [65, 100] corresponds to leveling a graph $G = (V, E)$ on $|V|$ levels with width one such that the span of G is minimized. We used this problem to prove some $\mathcal{NP}$-hardness results. In the periodic event scheduling problem [134] the vertices are positioned in a periodic time frame such that the span of each edge satisfies a minimum and maximum given value. Note that there a no discrete levels. In the cyclic job shop problem [74] several tasks are combined to jobs. The problem is to find a cyclic scheduling with fixed width such that each job is executed. For two tasks of the same job a maximal delay is defined. All these problems are related to the cyclic leveling problem. Hence, further research is needed to determine whether existing heuristics or approximation ratios can be transferred.

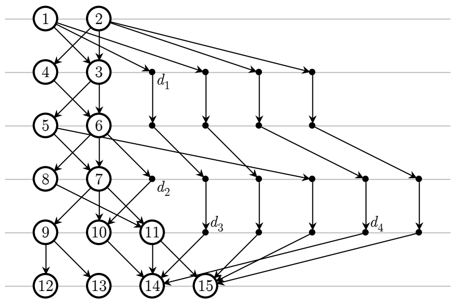
4 Crossing Reduction

In this chapter, we examine the crossing reduction phase, where the vertices on each level are permuted to minimize the overall number of crossings. See Figure 4.1 and Figure 4.2 as examples for the hierarchical and cyclic crossing reduction, respectively. Note that already the relative order of vertices on their respective levels determine the number of crossings, i. e., the x-coordinates of the vertices are not yet needed. Thus, crossing reduction is a combinatorial and not a geometric problem. We first give an overview of the hierarchical crossing reduction problem as shown in textbooks on graph drawing [32, 93, 137].

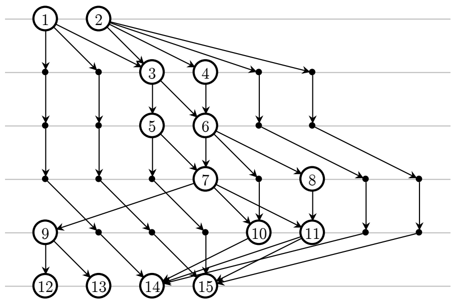
4.1 Hierarchical Crossing Reduction

Minimizing the overall number of crossings is $\mathcal{NP}$-hard, even if the graph has only two levels [70]. The traditional approach in the hierarchical Sugiyama framework for crossing minimization is a top-down and bottom-up level-by-level sweep algorithm on all levels. During a single step of one sweep, the crossings between two consecutive levels are minimized. This temporary solution has been the de facto standard crossing minimization method of the Sugiyama algorithm for nearly 30 years.

Starting with an arbitrary permutation of the vertices on the first level, the crossings between two levels V_{i-1} and V_i are subsequently minimized for $i = 2, \ldots, k$, where, at each step, the permutation of the upper level V_{i-1} is assumed to be fixed. Already the crossing minimization between two consecutive levels with one fixed, called *one-sided 2-level crossing minimization problem*, is $\mathcal{NP}$-hard [49, 69] and has since received much attention as discussed in the following paragraphs. However, the approximation ratios that have been obtained for the two-level crossing minimization do not carry over to the case of k-level graphs. Moreover, considering the crossings on all levels, Bastert and Matuszewski claim in [93] that the results of the 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” [93].

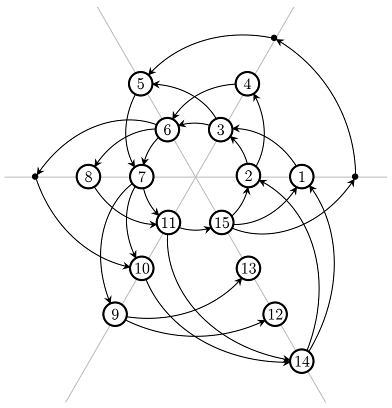


(a) Level graph G of G^*

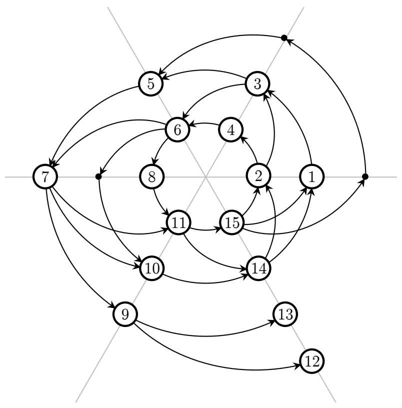


(b) Crossing reduction of G

Figure 4.1. Hierarchical crossing reduction



(a) Cyclic level graph G of G^*



(b) Crossing reduction of G

Figure 4.2. Cyclic crossing reduction

An important feature of crossing reduction algorithms is the avoidance of type 2 conflicts, which are crossings of inner segments. Among others, the standard fourth phase algorithm by Brandes and Köpf [19] assumes the absence of type 2 conflicts to align long edges vertically. Since their absence is one of the major aesthetic criteria for easily readable hierarchical drawings [93] and as it is dependent of the crossing reduction results, we support this property in our third phase.

Note that the number of crossings that all incoming segments of two vertices $u, v \in V_i$ have with each other only depends on the relative order of u and v on level i. Let the *crossing number* $c_{u,v}$ [44, 152] be the number of crossings of such segments if u is left of v.

The penalty minimization method [140] constructs a new graph with $|V_i|$ vertices. An edge $e = (u, v)$ is added whenever $c_{u,v} < c_{v,u}$. The edge is weighted with $c_{v,u} - c_{u,v}$. The edge $e = (u, v)$ implies fewer crossings if u is placed left of v, where the weight is the number of additional crossings if u is put right of v. Finding a weight-minimal feedback arc set of the penalty graph gives an optimal ordering of the vertices of V_i regarding the number of crossings. However, as the feedback arc set problem is $\mathcal{NP}$-hard, even for unweighted graphs [69, 92], heuristics have to be used [57]. The sum of $\min\{c_{u,v}, c_{v,u}\}$ for each pair of vertices $u, v \in V_i$ gives a trivial lower bound of the number of crossings. Jünger and Mutzel [89] state that this bound is surprisingly good, however.

There are several sorting heuristics [44] based on the crossing numbers $c_{u,v}$. The greedy-switch heuristic is based on bubblesort and swaps two neighboring vertices u and v if $c_{v,u} < c_{u,v}$. This heuristic is well suited as a postprocessing as it changes an existing permutation only if the number of crossings is reduced therewith [66]. The split heuristic is based on quicksort [80] and places a vertex v left or right of a pivot vertex u depending on whether $c_{v,u} < c_{u,v}$ or $c_{v,u} > c_{u,v}$.

The fast and simple barycenter [140] and median [51] heuristics place each vertex $v \in V_i$ in the barycenter or median position of its predecessors in V_{i-1}. Afterwards, V_i is sorted by these positions. Intuitively, on these positions segments are short and generate few crossings. Both heuristics find a permutation which gives a planar drawing if there exists one and both avoid type 2 conflicts. Note that planarity cannot be guaranteed for k-level graphs using barycenter or median. Furthermore, the median heuristic yields a 3-approximation of the optimal number of crossings for the one-sided 2-level crossing minimization.

Jünger and Mutzel [89] propose an ILP by means of which it is possible to compute the optimal permutation of V_i that can be used for instances with up to 60 vertices in V_i.

Mutzel and Weiskircher [113] suggest a planarization approach and remove a small number of segments to find a maximal planar subgraph in the 2-level graph which is drawn without crossings. The removed segments are inserted again and may cause many crossings. However, the resulting drawings are claimed to be easily understandable.

If the graph to be drawn is rather dense, i. e., it has much more segments than vertices, many crossings are inevitable when drawing each segment. Paulish [117] proposes to draw complete bipartite subgraphs in a special way: All segments of the subgraph are deleted, a new level between V_{i-1} and V_i with a new vertex v is inserted and all vertices of the complete bipartite subgraph are connected to v. Therefore, each original segment (u, w) of the bipartite subgraph is represented by the path u, v, w and the resulting drawing of the subgraph is plane then. However, this may lead to many new levels and can only be used for complete bipartite subgraphs.

Sifting was introduced as a heuristic for vertex minimization in ordered binary decision diagrams [127] and later adapted for the one-sided 2-level crossing reduction problem [104]. The idea is to keep track of the number of crossings while moving a vertex $v \in V_i$ along a fixed ordering of all other vertices in V_i. Then, v is placed to its locally optimal position. This is repeated several times for every vertex of V_i.

Several other heuristics include the greedy insertion [44], the stochastic heuristic [38], and the assignment heuristic [25]. Meta heuristics like genetic algorithms [103], tabu search [96], and the greedy randomized adaptive search procedure [95] were successfully applied to the one-sided 2-level crossing reduction problem.

4.1.1 Hierarchical k-level Crossing Reduction

Although, 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. Consequently, these heuristics push the crossings downwards or upwards to the next level. They are resolved at level k or 1, respectively as no next level exists in this case. An alternative is *centered 3-level crossing reduction* [32], 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. We propose a *centered 3-level sifting* heuristic. However, the centered 3-level approaches may generate type 2 conflicts.

All these algorithms take a local viewpoint, i.e., only the crossings between neighboring levels are investigated at a certain time. However, since the local approaches are agnostic to the overall structure of the graph, these approaches usually lead to rather poor results concerning the overall number of crossings incorporating all levels. To avoid such problems, approaches that take more or even all levels at a time into account have to be developed.

Tutte's algorithm [50] is the first global approach for hierarchical crossing reduction and is similar to the barycenter crossing reduction. First, the positions of the vertices on the extreme levels are fixed in a more or less arbitrary order. On each other level, the x-coordinate of a vertex is chosen as weighted average of the x-positions of all of its neighbors.

Another approach that, to some extent, also takes a global viewpoint and uses sifting, has been introduced by Matuszewski et al. [104], which we call *ordered k-level sifting* here to avoid confusion. They perform a sifting step for each vertex of the graph, where the vertices are processed in decreasing and then in increasing order of their degree. Hence, 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. In this approach, when investigating a vertex, all of its neighbors on both neighboring levels are considered. This is similar to our approach of centered 3-level sifting, where we consider the vertices level-by-level instead of ordered by degree.

Jünger et al. [85] presented an exact ILP approach for the $\mathcal{NP}$-hard k-level crossing minimization for small graphs.

The planarization approach [113] can be used in the k-level case as well. Chimani et al. [28] take that idea one step further and combine the leveling and crossing reduction phase and try to find a maximal upward planar subgraph, which is a DAG that can be drawn plane with all edges pointing in the same direction. Such a drawing can be used then to determine the leveling and the removed edges are inserted again. Utech et al. [149] use an evolutionary algorithm to combine the leveling and crossing reduction phases.

4.2 Cyclic Crossing Reduction

The cyclic crossing minimization problem is $\mathcal{NP}$-hard as well as it reduces from the hierarchical k-level crossing minimization problem.

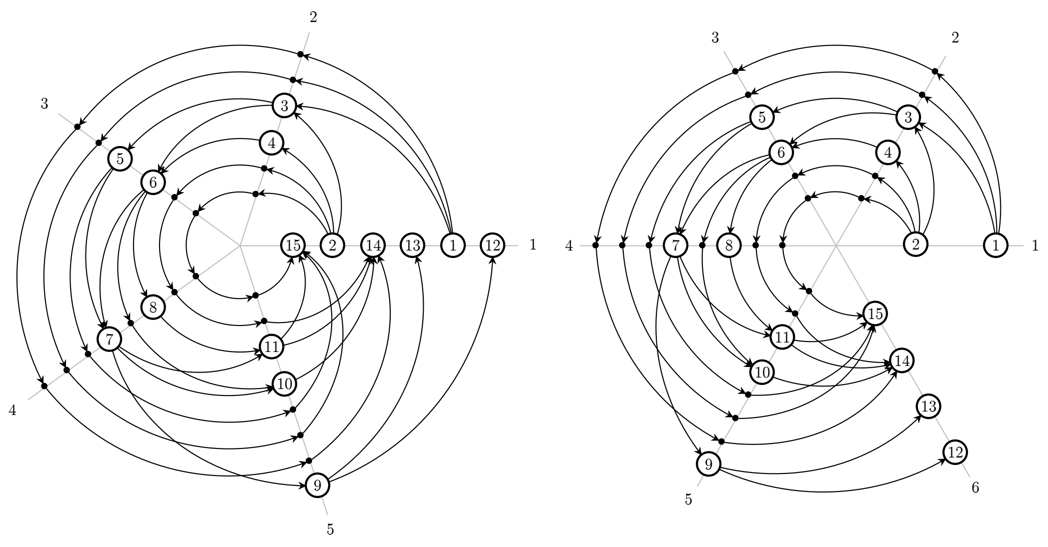
Theorem 4.1. *Let $G = (V, E, \phi)$ be a proper cyclic level graph and $l \in \mathbb{N}$. The problem whether there exist permutations of the vertices of each level such that the overall number of crossings is at most l is $\mathcal{NP}$-complete.*

Proof. The problem is obviously in $\mathcal{NP}$. To prove $\mathcal{NP}$-hardness, we give a reduction of the hierarchical crossing minimization problem, which is $\mathcal{NP}$-hard even for two levels [70]. Let $G_h = (V_h, E_h, \phi_h)$ be a proper (acyclic) k-level graph. We construct the cyclic $(k-1)$-level graph $G = (V_h, E_h, \phi)$ with $\phi(v) = \phi_h(v)$ if $\phi_h(v) < k$ and $\phi(v) = 1$ if $\phi_h(v) = k$. Then, all vertices of G_h of level 1 and k are on level 1 of G and all other levels remain unchanged. Note that on level 1 only sources and sinks are placed. The sinks can be ignored when treating the crossings between levels 1 and 2. Similarly, the sources can be ignored when treating the crossings between levels $k-1$ and 1. Hence, on level 1 only the relative order of the sources and the relative order of the sinks are important. We prove that there exist permutations of the vertices of G with at most l crossings if and only if such permutations exist for G_h.

"$\Rightarrow$": Let G have fixed permutations of vertices on each level which result in at most l crossings. Then we construct the permutations of G_h in the following way: The ordering of the vertices on level 1 is the relative order of all sources of level 1 in G, the ordering of the levels 2 to $k-1$ is taken directly from G and the ordering of level k is the relative order of all sinks of level 1 in G. Then G_h has the same number of crossings as G and, hence, at most l crossings.

"$\Leftarrow$": Let the permutations of G_h be fixed now with at most l crossings. The permutations of the levels 2 to $k-1$ can be taken directly. For the permutation of level 1, we concatenate the permutations of level 1 and k of G_h. Then G has obviously at most l crossings. □

See Figure 4.3(a) for an example of the transformation of the 6-level graph of Figure 4.1(b). When permuting level 1, only the relative order of the vertices 1 and 2 has an influence on the crossings between levels 1 and 2 and only the relative order of the vertices 12 to 15 influences the crossings between levels 5 and 1. Note that an even simpler reduction is possible: It transforms a k-level graph to a cyclic k-level



(a) As in the proof of Theorem 4.1 on 5 levels (b) Variant on 6 levels

Figure 4.3. Transformation of the 6-level graph of G^* (see Figure 4.1(b)) to cyclic level graphs

graph without changing the leveling at all (see Figure 4.3(b)). Then all permutations can be kept when switching between the hierarchical and cyclic case. However, the proof would not include the (degenerated) case of having only one cyclic level. Note that the ILP approach by Jünger et al. [85] can be used for cyclic k-level crossing minimization in a straight forward way by including additional constraints for the edges between the last and the first level.

Concerning cyclic crossing reduction, the level-by-level sweep procedures have a huge disadvantage. As there is no top or bottom level, the crossings are pushed to the next level after every step and potentially never resolved. But at some time, the algorithm has to terminate and leaves many crossings below or above the level that was permuted last. Hence, a completely new, more global approach is needed in the cyclic case.

In this chapter, we introduce a new global crossing reduction that avoids type 2 conflicts. The idea is to treat each vertex and each maximum connected subgraph consisting of dummy vertices as a block and perform sifting on all these blocks at the same time independently of their levels. Out experiments show that the algorithm gives better results than traditional sifting heuristics in the hierarchical case. It is easily applicable to other problems and has quadratic time complexity independent of the number of dummy vertices. We first give some preliminaries in Section 4.3 and then present our new global sifting heuristic in Section 4.4. To keep the description simple, we present the hierarchical version of the algorithm only. However, for the cyclic case no definition, algorithm, or description has to be changed, apart from replacing “level graph” by “cyclic level graph” at all occasions. In Section 4.5, we use the main idea of our global approach to improve the barycenter and median heuristic as well. We show how to apply the global sifting to related problems in Section 4.6. Finally, we give empirical results in Section 4.7, a summary in Section 4.8, and open problems in Section 4.9.

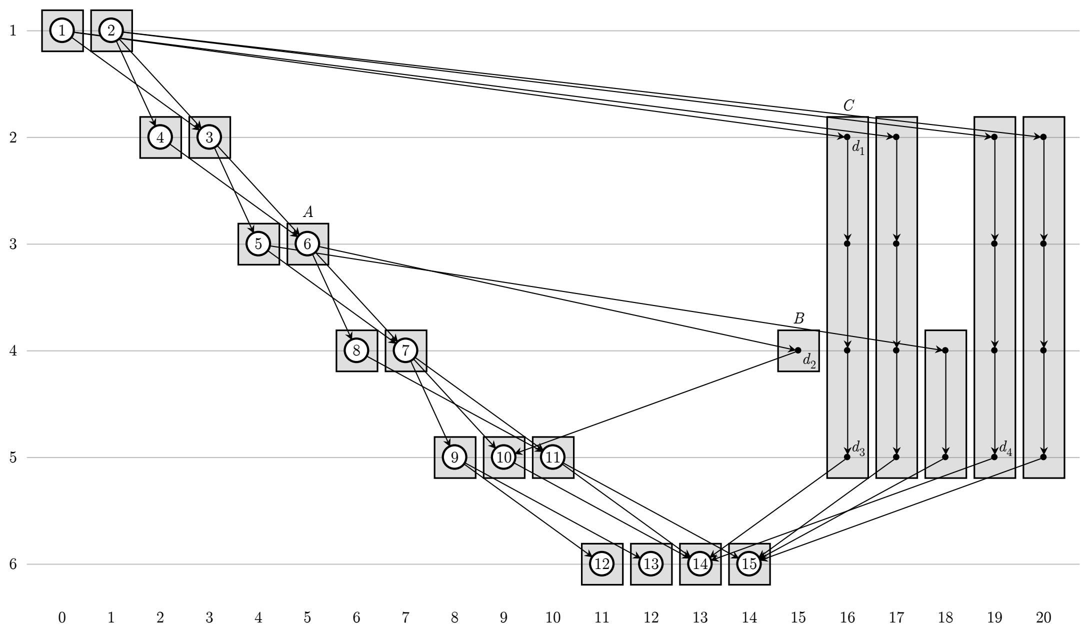
4.3 Preliminaries

We suppose that a directed graph without self-loops has been processed by the decycling and leveling phases. The outcome is an (acyclic) k-level graph. Let $G^p = (V^p, E^p, \phi^p)$ denote the proper version of G.

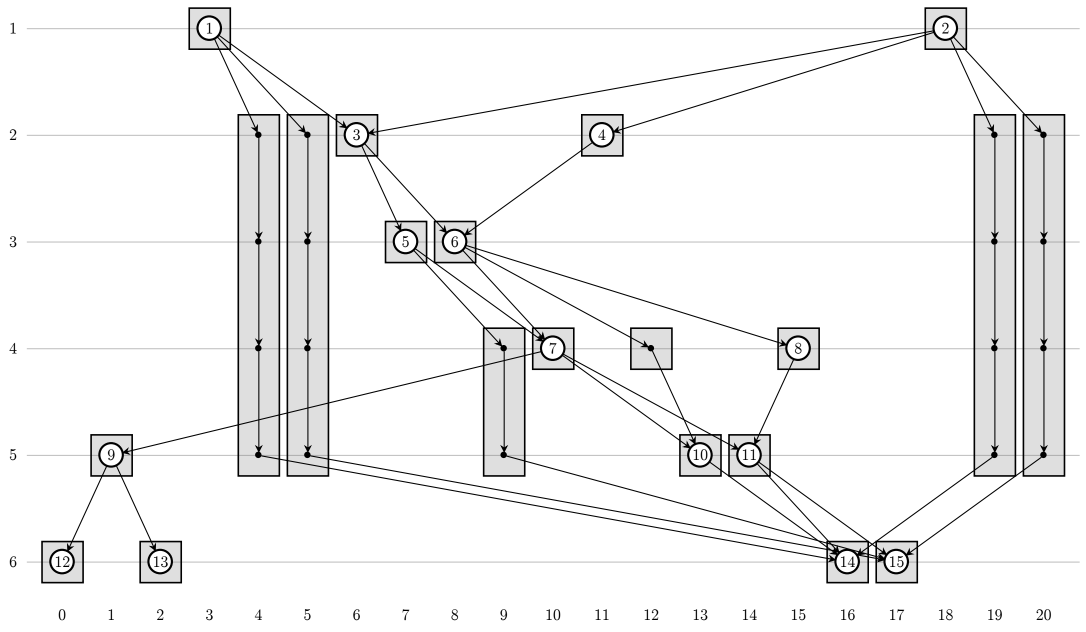
For a vertex $v \in V^p$ we denote the set of neighbors attached to incoming and outgoing segments by $N^-(v) := \{\, u \in V^p \mid (u, v) \in E^p \,\}$ and $N^+(v) := \{\, w \in V^p \mid (v, w) \in E^p \,\}$, respectively. In an ordered proper level graph, the vertices on each level as well as the 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.

Now, we define blocks: 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. As our algorithm processes blocks only the running time is then independent of the number of dummy vertices. For simplicity, we denote (dummy) vertices by lower case letters and blocks by upper case letters. For a block A, we define $x = \text{top}(A)$ ($y = \text{bottom}(A)$) to be the unique vertex x in A (y in A) with no incoming (outgoing) segments in A. The vertices $\text{top}(A)$ and $\text{bottom}(A)$ are the topmost and lowest vertex of the block A, respectively, which always exist but may be identical. We define $N^-(A) := N^-(\text{top}(A))$, $N^+(A) := N^+(\text{bottom}(A))$, $\deg(A) := |N^-(A)| + |N^+(A)|$, and the set of all level numbers on which A has (dummy) vertices as $\text{levels}(A)$. With $\text{block}(v)$ we denote the block of the vertex $v \in V^p$. 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. We call a single swap of two neighboring blocks in $\mathcal{B}$ a *sifting swap*, testing all positions for one block a *sifting step*, and the execution of a sifting step for every block in $\mathcal{B}$ a *sifting round*.

Figure 4.4(a) shows a possible result of the block building of the graph G in Figure 4.1(a) with each of the resulting 21 blocks having its own position π between 0 and 20. Each of the original 15 vertices forms its own block. Additionally, each of the 6 long edges, which were split by dummy vertices in Figure 4.1(a) builds one block in Figure 4.4(a). The blocks represent the vertices of a graph related to G^p, where the edges are the outer segments only. Block A in Figure 4.4(a) consists of one original vertex only and $\text{top}(A)$ as well as $\text{bottom}(A)$ is the original vertex 6. Furthermore, $N^-(A) = (4, 3)$, $N^+(A) = (8, 7, d_2)$, $\text{levels}(A) = \{3\}$, $\deg(A) = 2 + 3 = 5$ and $\pi(A) = 5$. Blocks B and C consist of dummy vertices only. Block B consists of one dummy vertex and has similar properties as block A. For block C $\text{top}(C) = d_1$,



(a) Blocks of G of Figure 4.1(a)



(b) Possible Result of the Crossing Reduction as in Figure 4.1(b)

Figure 4.4. Crossing Reduction using blocks

Algorithm 4.1. globalSifting

Input: Proper k-level graph $G^p = (V^p, E^p, \phi^p)$, number ρ of sifting rounds
Output: Graph G^p with vertices ordered by values $\pi(v)$ for each $v \in V^p$

```
1 create list $\mathcal{B}$ of all blocks in $G^p$
2 for $1 \leq i \leq \rho$ do
3     foreach $A \in \mathcal{B}$ do
4         $\mathcal{B} \leftarrow$ siftingStep($G^p$, $\mathcal{B}$, $A$)
5 foreach $v \in V^p$ do $\pi(v) \leftarrow \pi(\text{block}(v))$
6 return $G^p$
```

bottom$(C) = d_3$, $N^-(C) = (1)$, $N^+(C) = (14)$, levels$(C) = \{2, 3, 4, 5\}$, deg$(C) = 1+1 = 2$, and $\pi(C) = 16$. Note that blocks only consisting of dummy vertices always have degree two with one incoming and one outgoing edge. Figure 4.4(b) shows a different permutation of all blocks which is a potential result of our crossing reduction and corresponds to the permutation of each level of the graph in Figure 4.1(b).

4.4 Global Sifting

A major drawback of the established crossing reduction algorithms is their local view. We present a new approach using ideas from [19] and [52][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 sifting step. This eliminates the problem 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 4.1). At any time of the algorithm, $\pi(A)$ and $\phi(v)$ can be interpreted as the x- and y-coordinate of a vertex v in block A, respectively, to obtain a drawing that respects the current ordering of $\mathcal{B}$. All vertices of a block get the same x-coordinate and, hence, the ordering is type 2 conflict free. This is an important invariant of Algorithm 4.1.

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 the position with the fewest crossings. This is done sequentially for each block $A \in \mathcal{B}$ and repeated a certain number of rounds ρ. Our benchmarks in Section 4.7 indicate that ten rounds are sufficient in practice. Finally, each vertex is set to the position of its block (line 5) and the graph is returned.

[1]The authors of [52] 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.

4.4.1 Building the Block List

The first part of our global sifting algorithm is to partition the graph into blocks. Each block A gets an arbitrary but unique position $\pi(A)$ in the block list $\mathcal{B}$.

If the algorithm is only used to improve a given ordering in a postprocessing step, a straightforward initialization strategy is to topologically sort [30] the blocks according to the orderings on the levels from left to right in $\mathcal{O}(|E^p|)$. Note that using an existing ordering is only possible if the ordering is type 2 conflict free. As two blocks never cross, no ordering of $\mathcal{B}$ can represent a type 2 conflict. In Figure 4.4(a), the blocks of G are ordered such that the order corresponds to the permutations of the graph in Figure 4.1(a). Our experiments in Section 4.7 show, that a good initial ordering of the blocks in not important as the first sifting round has the highest impact on the number of crossings and changes the permutations of the vertices completely.

4.4.2 Initialization of a Sifting Step

To improve the performance of one sifting step [11] we 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 $I^-(A) = I^-(\text{top}(A))$ and $I^+(A) = I^+(\text{bottom}(A))$ of lengths $|I^-(A)| := |N^-(A)|$ and $|I^+(A)| := |N^+(A)|$, respectively. $I^-(A)$ stores the indices where $\text{top}(A)$ is stored in each adjacent block B's adjacency $N^+(B)$. More precisely, let $b = N^-(A)[i]$ be a neighbor of $\text{top}(A)$ with $B = \text{block}(b)$. Then $I^-(A)[i]$ holds the index at which $\text{top}(A)$ is stored in $N^+(B) = N^+(b)$. Symmetrically, $I^+(A)$ stores the indices at which $\text{bottom}(A)$ is stored in the adjacency $N^-(B)$ of each adjacent block B.

See Figure 4.4(a) for an example. There, $N^+(C)[0] = N^+(d_3)[0] = 14$ and $N^-(14) = (10, 11, d_3, d_4)$. $I^+(C)[0] = I^+(d_3)[0]$ then contains the position of d_3 in $N^-(14)$. Consequently, $I^+(C)[0] = I^+(d_3)[0] = 2$ as $N^-(14)[2] = d_3$.

The creation of the four arrays for each block (line 2 of Algorithm 4.3) can be done in $\mathcal{O}(|E|)$ time as Algorithm 4.2 shows: Traverse all blocks A in the current order of $\mathcal{B}$ and add $\text{top}(A)$ ($\text{bottom}(A)$) to the next free position j of the cleared adjacency array $N^+(\text{bottom}(B))$ ($N^-(\text{top}(B))$) 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. Therefore, we cache the first array position j as an attribute p of s.

Creating these arrays from scratch each time is not needed, however. At the end of one sifting step the block to sift is placed at its locally optimal position. After that, the next block is sifted and placed at the first position as the initialization. Hence, only the adjacencies of the neighbors of these two blocks have to be updated. This update gives a considerable speed up in practice and our implementation uses this idea. However, the theoretical running time is unaffected by this improvement. Hence, we skip the details and refer to [83].

Algorithm 4.2. sortAdjacencies

```
    Input: Proper $k$-level graph $G^p = (V^p, E^p, \phi^p)$, ordered list $\mathcal{B}$ of blocks in $G^p$
    Output: Ordered sets $N^\cdot(A)$ and $I^\cdot(A)$ for each block $A \in \mathcal{B}$
1   for $i \leftarrow 0$ to $|\mathcal{B}| - 1$ do
2       $\pi(\mathcal{B}[i]) \leftarrow i$
3       clear arrays $N^-(\mathcal{B}[i])$, $N^+(\mathcal{B}[i])$, $I^-(\mathcal{B}[i])$ and $I^+(\mathcal{B}[i])$
4   foreach $A \in \mathcal{B}$ do
5       foreach $s \in \{ (u,v) \in E^p \mid v = \mathrm{top}(A) \}$ do
6           add $v$ to the next free position $j$ of $N^+(u)$
7           if $\pi(A) < \pi(\mathrm{block}(u))$ then
8               $p[s] \leftarrow j$                          // first traversal of $s$
9           else
10              $I^+(u)[j] \leftarrow p[s]$                  // second traversal of $s$
11              $I^-(v)[p[s]] \leftarrow j$
12      foreach $s \in \{ (w,x) \in E^p \mid w = \mathrm{bottom}(A) \}$ do
13          add $w$ to the next free position $j$ of $N^-(x)$
14          if $\pi(A) < \pi(\mathrm{block}(x))$ then
15              $p[s] \leftarrow j$                          // first traversal of $s$
16          else
17              $I^-(x)[j] \leftarrow p[s]$                  // second traversal of $s$
18              $I^+(w)[p[s]] \leftarrow j$
```

4.4.3 Sifting Step

In a sifting step (Algorithm 4.3), 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 causes the least number of crossings. Note that it is not necessary to count the crossings for each position of A. As in [11] 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).

4.4.4 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, during the computation of the change in the number of crossings, our global approach takes the whole neighborhood of both blocks into account. Lemma 4.1 states which segments are involved.

Lemma 4.1. *Let $\mathcal{B}$ be the block list in the current ordering. Let $B \in \mathcal{B}$ be the*

Algorithm 4.3. siftingStep

```
Input: Proper $k$-level graph $G^p = (V^p, E^p, \phi^p)$, ordered list $\mathcal{B}$ of blocks in $G^p$,
       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 sortAdjacencies($G^p$, $\mathcal{B}'$)
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     $\chi \leftarrow \chi +$ siftingSwap($A, \mathcal{B}'[p]$)
7     if $\chi < \chi^*$ then
8         $\chi^* \leftarrow \chi$; $p^* \leftarrow p$
9 return $\mathcal{B}'[1] \prec \cdots \prec \mathcal{B}'[p^*] \prec A \prec \mathcal{B}'[p^* + 1] \prec \cdots \prec \mathcal{B}'[|\mathcal{B}'| - 1]$
```

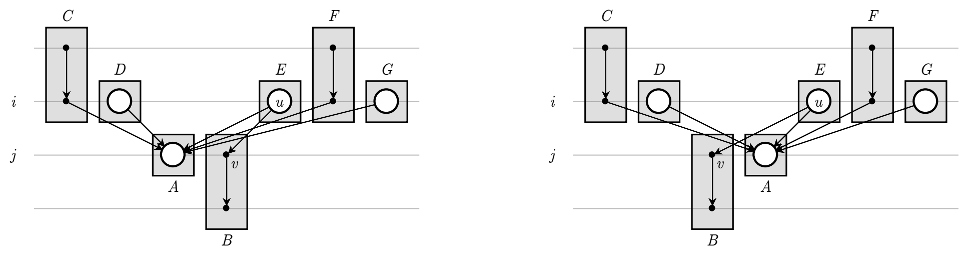
successor of $A \in \mathcal{B}$. If swapping A and B changes the crossings between any two segments, then one of them is an incident outer segment of A or B. The other segment is an incident outer segment of the same kind (incoming or outgoing) of the other block or an inner segment of the other block.

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 a and c were not, $\phi(b) = \phi(d)$ is the upper level of A or B and 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. Then, one segments is an outgoing outer segment of A or B. The second segment is either an outgoing outer segment or an inner segment of the other block. □

Hence, at most four spaces between levels have to be treated when computing the change in the number of crossings while swapping two blocks. Actually, only two of these four spaces can have a change in the number of crossings in the hierarchical case. In the cyclic case, all four spaces can be affected, however.

Proposition 4.1. *Let $\mathcal{B}$ be the block list in the current ordering. Let $B \in \mathcal{B}$ be the successor of $A \in \mathcal{B}$ before swapping A and 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 4.1.*



(a) Before swapping A and B (b) After swapping A and B

Figure 4.5. Change of crossings when swapping A and B

See Figure 4.5 for an example. The incoming segment (u, v) of block B starts at block E. Thus, all incoming segments of A starting at a block left of block E, namely the segments starting at block C and D, cross (u, v) after the swap of A and B only. The segment connecting block E and A never crosses (u, v) and the incoming segments of A starting right of block E, namely the segments starting at block F and G, cross (u, v) before the swap only.

Algorithm 4.4 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 2–7). 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 [11] (lines 18–35): The neighbors are traversed from left to right. If a neighbor of a is found (lines 25–27) its segment will cross all remaining $s - j$ incident segments of b after the swap. If a neighbor of b is found (lines 28–30) its segment has crossed all remaining $r - i$ incident segments of a before the swap. With common neighbors both cases occur at the same time (lines 32–34). An update of the adjacency after a swap (line 12) is only necessary if a and b have common neighbors. Algorithm 4.5 shows how this can be done in overall $\mathcal{O}(\deg(A) + \deg(B))$ time similarly to the crossing counting function `uSwap` in Algorithm 4.4.

4.4.5 Time Complexity

This section gives the time complexity of our global sifting algorithm. For the proof we need the following lemma.

Algorithm 4.4. siftingSwap

```
Input: Consecutive blocks $A, B$
Output: Change in crossing count
begin
    $\mathcal{L} \leftarrow \emptyset$
    $\Delta \leftarrow 0$
    if $\phi(\text{top}(A)) \in \text{levels}(B)$ then $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\phi(\text{top}(A), -)\}$
    if $\phi(\text{bottom}(A)) \in \text{levels}(B)$ then $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\phi(\text{bottom}(A), +)\}$
    if $\phi(\text{top}(B)) \in \text{levels}(A)$ then $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\phi(\text{top}(B), -)\}$
    if $\phi(\text{bottom}(B)) \in \text{levels}(A)$ then $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\phi(\text{bottom}(B), +)\}$
    foreach $(l, d) \in \mathcal{L}$ do
        let $a$ be the vertex of block $A$ with $\phi(a) = l$
        let $b$ be the vertex of block $B$ with $\phi(b) = l$
        $\Delta \leftarrow \Delta + \mathsf{uSwap}(a, b, N^d(a), N^d(b))$
        updateAdjacencies$(a, b, N^d(a), I^d(a), N^d(b), I^d(b))$
    swap positions of $A$ and $B$ in $\mathcal{B}$
    $\pi(A) \leftarrow \pi(A) + 1$
    $\pi(B) \leftarrow \pi(B) - 1$
    return $\Delta$
end
function $\mathsf{uSwap}(a, b, N^d(a), N^d(b))$: 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 4.5. updateAdjacencies

```
Input: Vertices $a, b \in V^p$, $N^d(a), I^d(a), N^d(b), I^d(b)$
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 $I^{-d}(z)$
        $I^d(a)[i] \leftarrow I^d(a)[i] + 1$
        $I^d(b)[j] \leftarrow I^d(b)[j] - 1$
        $i \leftarrow i + 1$
        $j \leftarrow j + 1$
```

Lemma 4.2. *Let $G = (V, E, \phi)$ be a, not necessarily proper, level graph and $\mathcal{B}$ be its list of blocks. Then $\sum_{B \in \mathcal{B}} \deg(B) \leq 4|E|$.*

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 4.2. *One round of global sifting (Algorithm 4.1) has a time complexity of $\mathcal{O}(|E|^2)$ for a, not 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 with Algorithm 4.4. Initializing a sifting step by sorting the adjacency lists $N^\cdot(\cdot)$ and $I^\cdot(\cdot)$ in Algorithm 4.2 takes $\mathcal{O}(\sum_{B \in \mathcal{B}} \deg(B)) = \mathcal{O}(|E|)$ time. A sifting step of a block A has a time complexity of $\mathcal{O}(\sum_{B \in \mathcal{B} \setminus \{A\}} (\deg(A) + \deg(B))) = \mathcal{O}(|E| \cdot \deg(A))$. A sifting round positioning each block $A \in \mathcal{B}$ takes the time $\mathcal{O}(\sum_{A \in \mathcal{B}} (|E| \cdot \deg(A)) = \mathcal{O}(|E|^2)$. Since $|V^p| \leq |E| \cdot k \in \mathcal{O}(|E|^2)$ (no empty levels), traversing all (dummy) vertices in pre- and postprocessing has no effect on the worst case time complexity. □

The time complexity is independent of the number of dummy vertices in the graph. Traditional crossing reduction algorithms like barycenter or median have a (nearly) linear time complexity in the size of the graph. However, they process proper graphs only, which can have up to $\mathcal{O}(k \cdot |E|) \subseteq \mathcal{O}(|V|^3)$ dummy vertices. Therefore, the time complexity of $\mathcal{O}(|E|^2)$ is not as bad as it might seem at a first glance. This is also confirmed by the experimental results given in Section 4.7.

4.5 Simple Global Crossing Reductions

Using the idea of blocks, we also extend the simple barycenter and median crossing reduction strategies: We iteratively take the π-positions of the blocks in $\mathcal{B}$ and compute for each block the barycenter or median, respectively, and sort $\mathcal{B}$ according to these values. Our benchmarks in Section 4.7 show that both are very fast, however, are not competitive with global sifting in the number of crossings.

Theorem 4.3. *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 4.2. Sorting the barycenters takes $\mathcal{O}(|E|\log|E|)$ time. The medians can be sorted in $\mathcal{O}(|E|)$ time using bucket sort. □

Note, if the used coordinate assignment phase needs dummy vertices (which are unnecessary here), then a postprocessing in $\mathcal{O}(|V^p|) \subseteq \mathcal{O}(|E| \cdot k)$ time is needed to create the dummy vertices.

4.6 Applications of the Global Crossing Reduction

In this section, we show how to use the idea of blocks in several other established algorithms to improve their performance in a straight forward way. Further, this advances the drawability of their results as type 2 conflicts are avoided.

4.6.1 Radial Level Graphs

In a radial level drawing [2], the levels are concentric circles (see Figure 1.16(b)). These drawings visualize distances or importance and are the traditional drawings for social networks. They map structural centrality of the graph to geometric centrality. The vertices on each level are ordered in a cyclic way.

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 [2], 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 line 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 counter-clockwise and clockwise segments on the levels above and below A, see Figure 4.6. Since we can do this independently of each other and add the results of the change in crossings to Δ in Algorithm 4.4, we use the same technique as in [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 overall running time raises to $\mathcal{O}(|E|^3)$ due to the test whether changing the orientation of some of the first edges of the adjacency of A during the swap reduces the number of crossings. If this is the case, their offset is incremented and, hence, they are put last in the adjacency. The radial coordinate assignment phase in [2] relies on the obtained absence of type 2 conflicts.

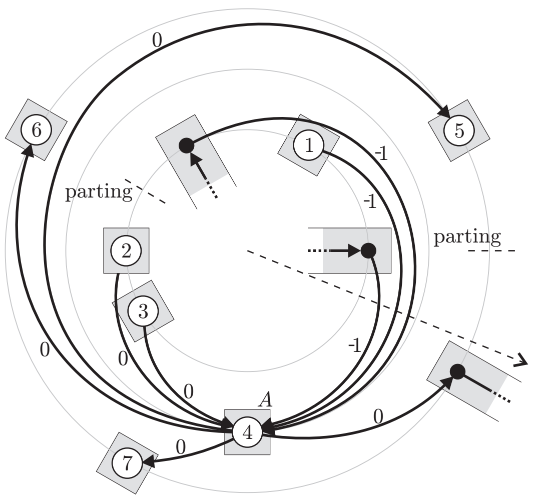


Figure 4.6. Partings of the block A in a radial drawing

4.6.2 Optimal Crossing Reduction Using an ILP

Jünger et al. [85] gave an ILP formulation for the exact crossing minimization of k-level graphs. Note that the optimal solution may contain type 2 conflicts. Using the idea of blocks we give a direct formulation which forbids type 2 conflicts using fewer variables. These can be excluded in the ILP by Jünger et al. by adding additional constraints, which results in a similar ILP after simplification. However, using blocks gives a more direct correspondence to the ILP.

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}. $x_{AB} = 1$ indicates 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 $\pi(A) < \pi(B) < \pi(C)$ which have at least one common level, i. e., $\text{levels}(A) \cap \text{levels}(B) \cap \text{levels}(C) \neq \emptyset$, we add the condition $0 \leq x_{AB} + x_{BC} - x_{AC} \leq 1$ to ensure that there are no 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 $A = \text{block}(a), \ldots, D = \text{block}(d)$. 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 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. According to our experiments, this approach is unfortunately only usable for graphs up to 40 vertices.

Obviously, a planar embedding of the level graph does not contain any type 2 conflicts. Consequently, our version of the ILP can be used to test level planarity as well as the original version.

4.6.3 Level Planarity Testing Using the Vertex Exchange Graph

Harrigan and Healy [76] use a data structure called *vertex exchange graph* as a simple test for level planarity in $\mathcal{O}(|V^p|^2)$ time. Using the idea of blocks, it is straight forward to improve the running time of the test of $\mathcal{O}(|V^p|^2)$, i. e., $\mathcal{O}((|V| \cdot k)^2)$ worst case, to $\mathcal{O}(|V|^2)$ similarly to the ILP above: Each pair of 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 minimize crossings. See Chapter 6.2 for a slightly longer description of these algorithms.

4.6.4 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 [60]. 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 iteratively. The time complexity is 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 clustering layer.

4.7 Experimental Results

In this section, we compare the practical performance of our global crossing reduction heuristics with other algorithms in the hierarchical and cyclic case. We use the following abbreviations in the figures: iterative one-sided 2-level barycenter (B), median (M), and sifting (S), iterative centered 3-level sifting (3S), ordered k-level sifting (OS), and global barycenter (GB), median (GM), and sifting (GS). For each of the former four heuristics, we applied ten top-down and bottom-up sweeps and for each of the latter ones we performed ten rounds. For each graph size, we have generated ten arbitrary graphs with an aspect ratio, i. e., maximum number of vertices per level vs. number of levels, of the *golden rectangle*, i. e., of $\frac{1+\sqrt{5}}{2}$. All benchmarks were run on a 2.8 GHz workstation under the Java 6.0 platform of Sun Microsystems, Inc.

4.7.1 Hierarchical Crossing Reduction

In Figure 4.7 the running times of the algorithms are shown. Clearly, the simple as well as the global barycenter and median heuristics are very fast due to their (nearly) linear time complexity. The sifting methods are slower as their time complexity is quadratic in the size of the subgraphs between two levels. Global sifting has an overall quadratic time complexity and is the slowest heuristic in the benchmarks. However, the running time of 10 seconds for a graph with 10000 (dummy) vertices is still practical. Figure 4.8 shows that the global sifting benefits from having many dummy vertices. The sifting heuristics do not differentiate between vertices and dummy vertices. Hence, increasing the percentage of dummy vertices does not change their running time much. However, in global sifting the number of blocks decreases as the number of dummy vertices increases. The running time even reaches the other sifting heuristics for graphs with 80% dummy vertices.

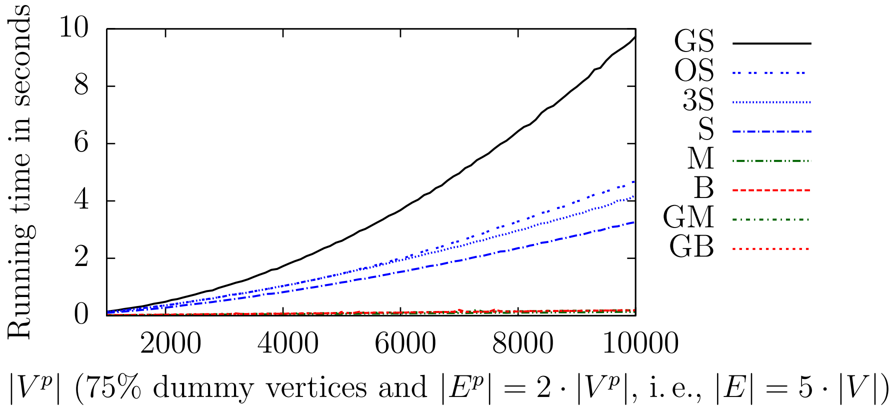


Figure 4.7. Benchmark: running times (hierarchical)

Figure 4.9 shows the reduction of the number of crossings compared to an arbitrary initial ordering. The number of type 2 conflicts for each heuristic is shown in Figure 4.10. Note that all missing algorithms guarantee the absence of type 2 conflicts. The global barycenter (GB) and median (GM) heuristics perform very poorly. The classic barycenter (B) and median (M) algorithms can eliminate nearly 60% of the crossings and produce no type 2 conflicts. Classical sifting (S) performs similarly but produces some type 2 conflicts. Iterative centered 3-level sifting (3S) and ordered k-level sifting (OS) remove more crossings but create up to 10000 type 2 conflicts. They permute one level while fixing both neighboring levels. Hence, they tend to keep type 2 conflicts. Global sifting eliminates about 66% of the crossings without any type 2 conflicts. Put differently, the sifting heuristics (3S, OS) produce nearly 10% more crossings and many type 2 conflicts. Comparing heuristics

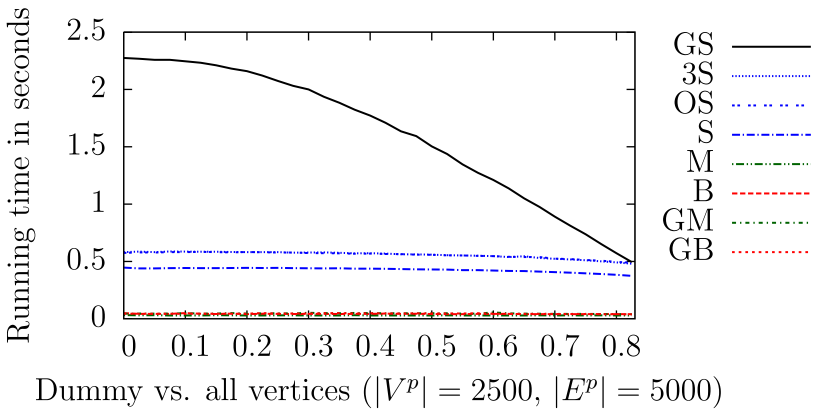


Figure 4.8. Benchmark: running times with different rates of dummy vertices (hierarchical)

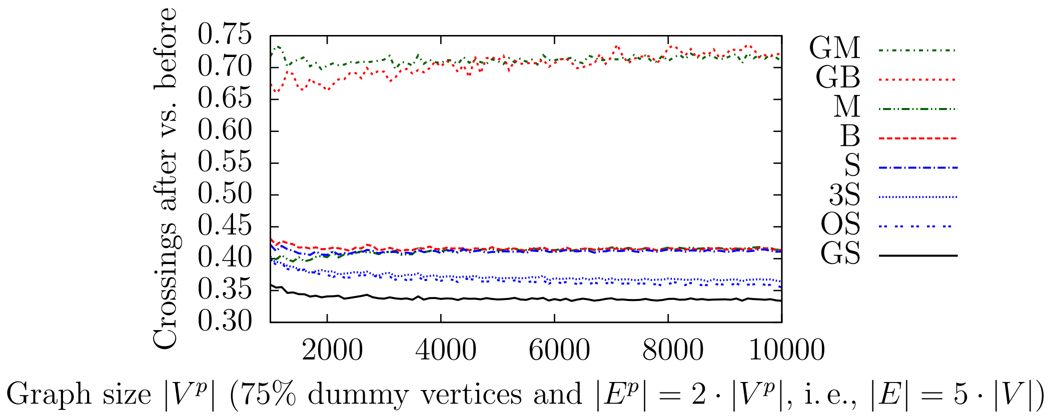


Figure 4.9. Benchmark: number of crossings after vs. before applying the crossing reduction (hierarchical)

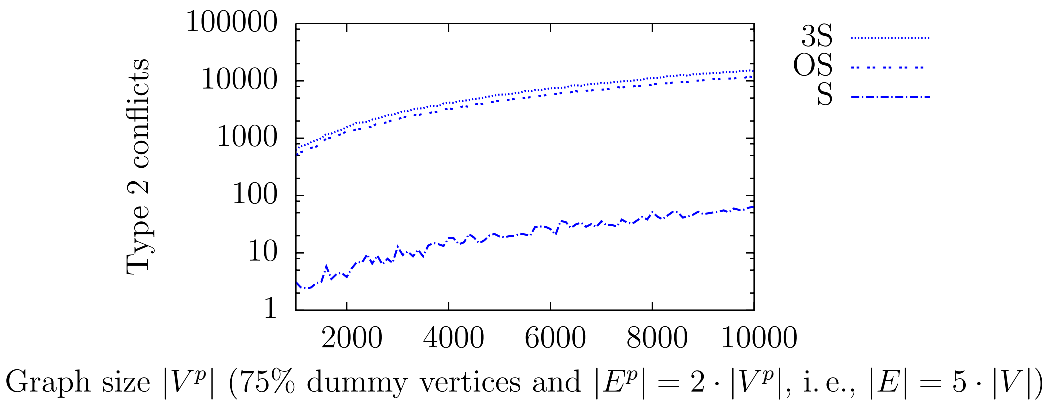


Figure 4.10. Benchmark: number of type 2 conflicts (hierarchical)

guaranteeing no type 2 conflicts only, barycenter and median generate nearly 25% more crossings. These results would improve even more when comparing against an optimal solution or a lower bound. Note that an optimal solution without type 2 conflicts has more crossings than an optimal solution with type 2 conflicts in general. However, the optimal solution can be found in practical time for very small graphs only. In Figure 4.11 we compare the heuristics with the optimal solution for graphs with up to 35 vertices. Although our implementation permits to exclude type 2 conflicts explicitly, the optimal solutions for these graphs never included any type 2 conflicts in the first place. The results should not be over-interpreted. However, they indicate that the difference in the number of crossings between global sifting and other heuristics when compared to the optimal solution could be higher than 25%.

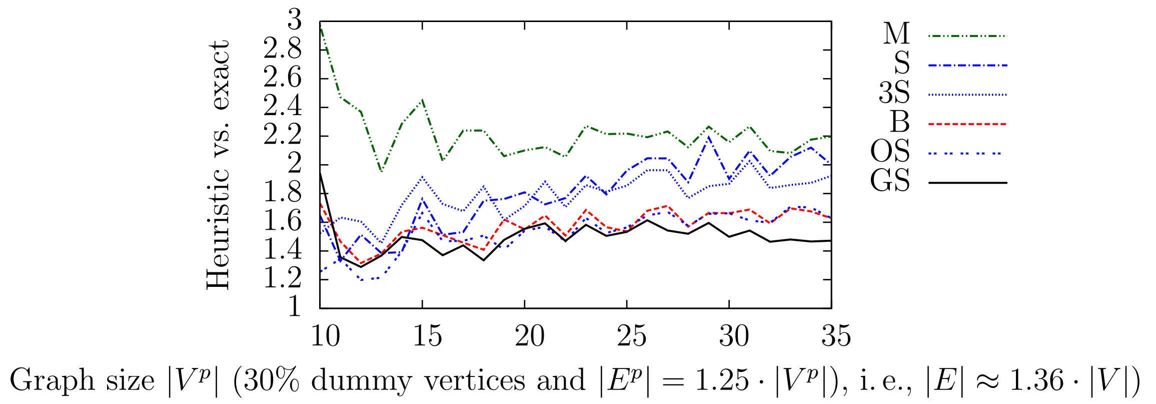


Figure 4.11. Benchmark: number of crossings compared to optimal solutions (hierarchical)

4.7.2 Cyclic Crossing Reduction

As we stated before, level-by-level sweeping algorithms do not perform well in the cyclic case as the algorithm has to terminate at some point and leaves many crossings between the last considered level and the next one. However, we compare our global heuristics with them anyway, as to our best knowledge, there are no other cyclic crossing reduction algorithms yet.

The results regarding the time complexity (see Figure 4.12) and the number of crossings (see Figure 4.13) are nearly identical to the hierarchical case. Figure 4.14 presenting the number of type 2 conflicts includes some changes, however. Now, the barycenter and median heuristics cannot guarantee the absence of type 2 conflicts any more. They produce up to 800 conflicts. Although barycenter and median cannot guarantee no type 2 conflicts in general in the cyclic case they still produce no type 2 conflicts between the currently permuted and the fixed neighboring level. Hence, all these conflicts are positioned between the last permuted level and the

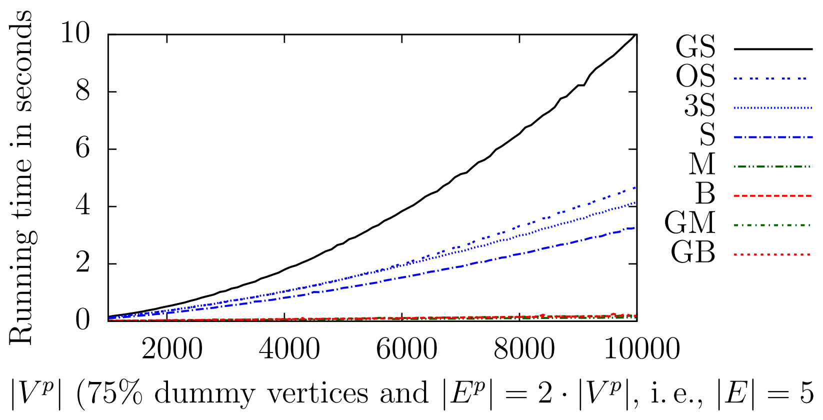


Figure 4.12. Benchmark: running times (cyclic)

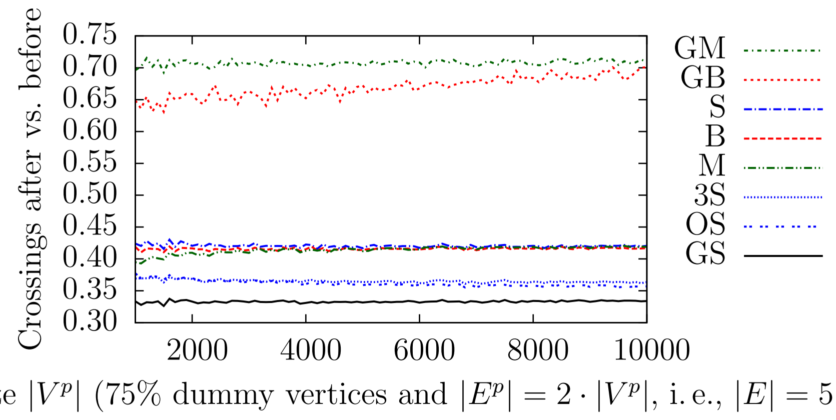


Figure 4.13. Benchmark: number of crossings after vs. before applying the crossing reduction (cyclic)

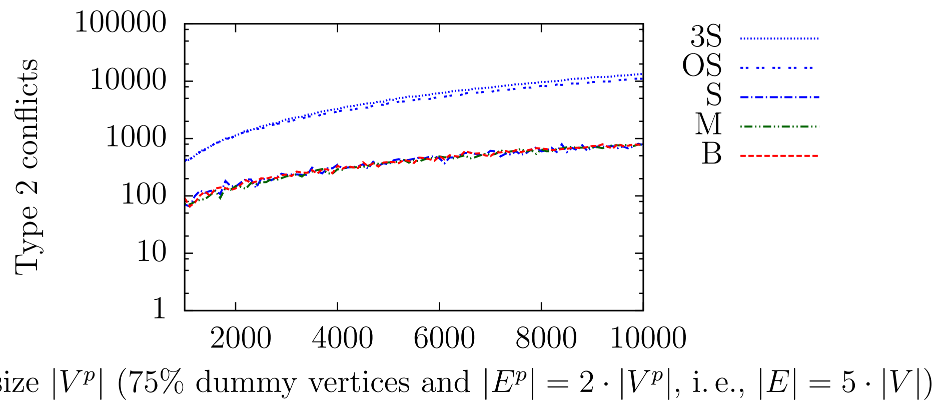


Figure 4.14. Benchmark: number of type 2 conflicts (cyclic)

next one. This shows that level-by-level sweeping heuristics are completely unsuited for the cyclic case. The sifting heuristic (S) performs similarly to the barycenter and median heuristics. The remaining heuristics (OS, 3S) generate much more type 2 conflicts in the cyclic as well.

Figure 4.15 and Figure 4.16 present the influence of the number of sweeps or rounds in the number of crossings. Again, global barycenter and median perform very poorly. Interestingly, the simple barycenter, median, and sifting heuristics seem to need one sweep only. The remaining heuristics (OS, 3S, GS) improve their results over the first five sweeps or rounds. As the two plots look nearly identical although the size of the input graphs differ by the factor five, we conclude that at least ten rounds of global sifting should be sufficient in practice.

The first round of global sifting removes nearly 60% of the crossings of the input graph. Hence, using another crossing reduction as preprocessing does not seem sensible.

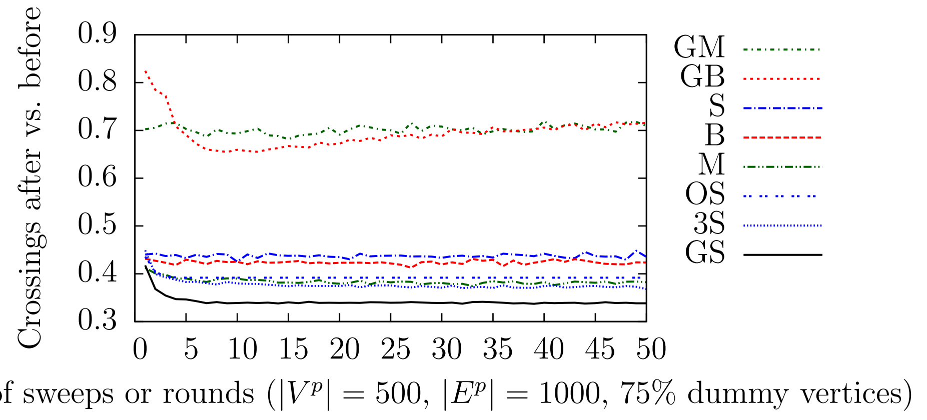


Figure 4.15. Benchmark: number of sweeps or rounds on graphs with 500 (dummy) vertices (cyclic)

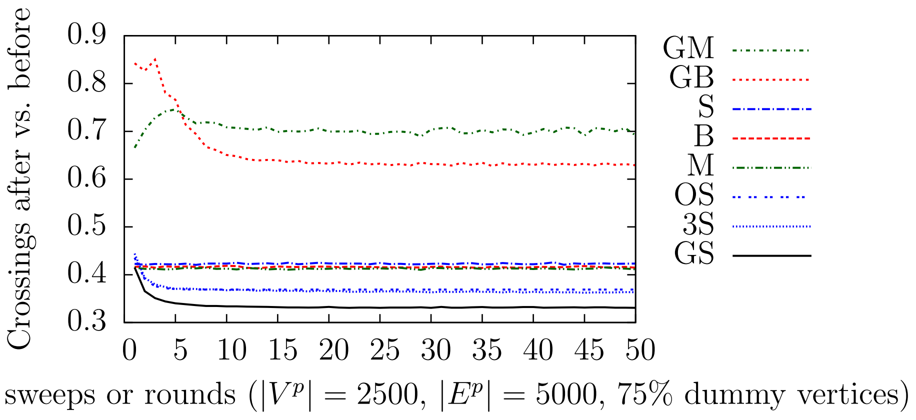


Figure 4.16. Benchmark: number of sweeps or rounds on graphs with 2500 (dummy) vertices (cyclic)

4.8 Summary

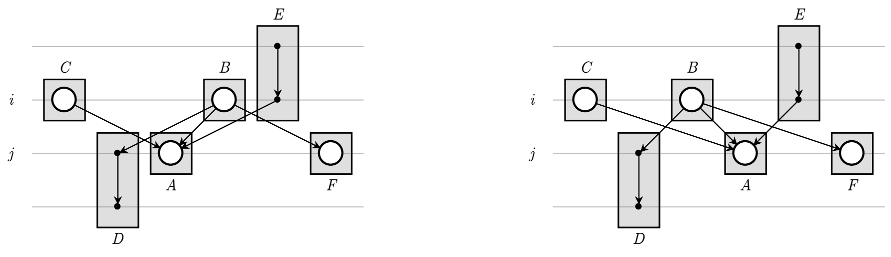
We presented the first constructive and practical high-quality heuristic in terms of few crossings, for crossing reduction in k-level graphs with a global view on long edges. This was an open problem since the introduction of the hierarchical framework [140] in 1981. We treat each original vertex and each subgraph consisting of dummy vertices only as one block. In the global sifting heuristic we iteratively search for the locally best position of each block. We investigated global barycenter and median variants as well. However, they give poor results.

In the hierarchical case, the global sifting heuristic gives nearly 10% less crossings than heuristics like ordered k-level sifting or centered 3-level sifting and additionally avoids type 2 conflicts. Comparing it with other heuristics, which guarantee to avoid type 2 conflicts as well, global sifting produces about 20% less crossings.

For cyclic and radial crossing reduction we presented the first algorithms which guarantee the absence of type 2 conflicts. Our approach can easily be used to simplify or improve several other algorithms concerning level planarity or crossing reduction.

4.9 Open Problems

Although the time complexity of our global sifting algorithm is reasonable as shown in our tests, the theoretical time complexity of $\mathcal{O}(|E|^2)$ is rather high. Traditional sweeping crossing reduction algorithms, like barycenter or median, have a (nearly) linear time complexity (in the size of the proper level graph) but cannot be used for cyclic crossing reduction. Therefore, a completely different approach for the cyclic case leading to (near) linear time complexity is of interest.



(a) Before swapping A and B (b) After swapping A and B

Figure 4.17. No change of crossings when swapping blocks A and B on disjoint levels

The practical time complexity of the global sifting algorithm could be improved by testing the positions of blocks which change a relative order on a level only. Often blocks on disjoint levels are swapped which does not change any permutation on any level. See Figure 4.17 for an example. The worst case situation is having a level with only one vertex v. The global sifting algorithm tests all $\mathcal{O}(|E|)$ positions for the block of v although each of them yields the same permutation of the level of v. Each such swap is computed in constant time. Nevertheless, avoiding unnecessary swaps is desirable, although it will not improve the theoretical worst case complexity: Consider a complete bipartite graph $G = (V_1 \cup V_2, E)$ with the vertices of V_1 on level 1 and the vertices of V_2 on level 3. Then, level 2 contains $\mathcal{O}(|E|) = \mathcal{O}(|V|^2)$ dummy vertices. For each block representing a dummy vertex, $\mathcal{O}(|E|)$ positions on level 2 have to be tested. Hence, having a time complexity of $\mathcal{O}(|E|^2)$ when avoiding unnecessary swaps is possible as well, even if $\mathcal{O}(|E|) = \mathcal{O}(|V|^2)$.

A rather simple approach to reduce the number of unnecessary swaps is to stop a sifting step if all adjacent blocks of the current block A are left of A already. Swapping A further to the right can only increase the number of crossings then. Similarly, the first position to test for a block A can be directly left of its leftmost adjacent block.

Merging several blocks would speed up the algorithm and could even reduce the number of crossings. Consider a sink with only one incoming edge. The edge and the sink are represented by two blocks. If the two blocks are neighbors in $\mathcal{B}$, moving one of them to a very different position will cause the outer segment between the two blocks to be a very long, nearly horizontal, line which potentially causes many crossings. However, if the edge and the sink are represented as one block, they would always be moved at the same time and, thus, a better position for both might be found. This could even be extended to unite all blocks of vertices and edges of a long path. However, the algorithm would then ensure, similarly to type 2 conflicts, that such long paths would not cross. In the cyclic framework it has to be ensured that such a block does not wrap around the center more than once. Apart from the block building, the algorithm would not have to be changed.

If a block causes the same number of crossings on several positions the length of its incident segments could be taken into account. Keeping these segments short could help avoiding local minima in the crossing reduction. Furthermore, shorter segments could reduce the width of the final drawing. Especially in cyclic drawings as long outer segments promote large cyclic dependencies in the left-to-right order of the vertices. These dependencies are the main new problem in the cyclic coordinate assignment phase in Chapter 5.3. Further research in necessary to determine whether reducing the length of outer segments while adding some crossings is sensible. The global sifting algorithm could easily use a term depending on the length of outer segments and the number of crossings to position a block.

The idea of blocks could possibly be used in several other problem settings. In incremental algorithms [16, 111, 112, 116, 132], drawings of graphs preserve their structure when new vertices or edges are added. The list of blocks could be used to find a suitable position for new vertices or a suitable routing for new long edges. A first approach to applying the blocks to the radial crossing reduction algorithm in [2] leads to a time complexity of $\mathcal{O}(|E|^3)$. Further research is needed to evaluate if a time complexity of $\mathcal{O}(|E|^2)$ can be achieved in the radial case as well.

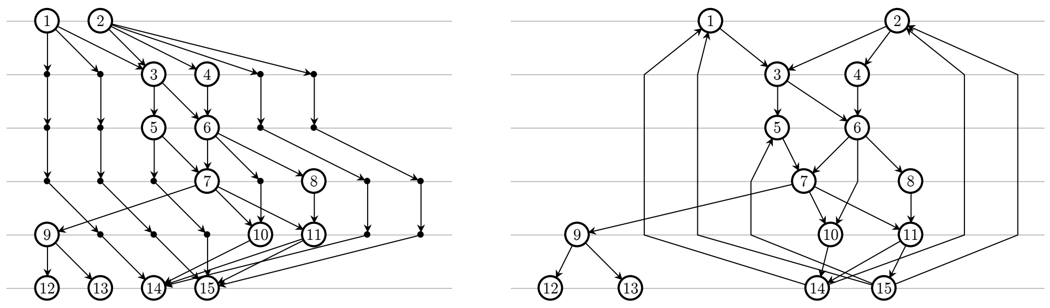
When dealing with vertices with arbitrary sizes [63, 132], the blocks might help as well. Note that the width of a vertex does not influence the crossing reduction phase. The height does, however, if large vertices span several levels. Here, a block spanning these levels could represent such a vertex easily. Special treatment of these blocks would be necessary to avoid crossings of such vertices with other edges, though.

5 Coordinate Assignment

The final phase of the Sugiyama framework is the coordinate assignment. In the hierarchical case, the x-coordinate for each vertex is determined. The y-coordinates are given implicitly by the leveling determined in the second phase. See Figure 5.1 for an example. Figure 5.1(a) shows the result of the crossing reduction phase for the example graph G^*. In Figure 5.1(b), the final drawing of G^* is shown. Some trivial postprocessing steps were already applied to that drawing.

In the cyclic Sugiyama framework, the coordinate assignment phase has to determine the distance of each vertex v from the center. The angle is given by the leveling. Figure 5.2(a) shows a possible result of the crossing reduction of G^* and Figure 5.2(b) depicts the final 2D drawing. In Figure 5.2(c), the drawing on a cylinder is shown and Figure 5.2(d) represents the surface of the cylinder, which is the intermediate drawing with dummy vertices already removed as well.

Important drawing conventions and aesthetic criteria for the coordinate assignment phase include aligning long edges, keeping the number of bends on long edges small, balancing vertices between their adjacent neighbors, and keeping the width of the drawing small. In the hierarchical case, long edges are aligned vertically. In the cyclic case, we try to align long edges on concentric circles. However, this is not always possible as this chapter shows, which is organized as follows: We give an overview over traditional hierarchical coordinate assignment algorithms in Section 5.1 and necessary definitions in Section 5.2. The main part of the chapter is the description of our cyclic coordinate assignment algorithm in Section 5.3. We analyze the time complexity and the area of the drawings in Section 5.4. Cyclic dependencies in the left-to-right order of the vertices are the main new problem in the cyclic coordinate assignment phase. We show how to avoid them in Section 5.5. Finally, we give a summary in Section 5.6, open problems in Section 5.7 and some example drawings in Section 5.8.



(a) Ordered level graph G of G^* (b) Hierarchical drawing of G

Figure 5.1. Hierarchical coordinate assignment and postprocessing

5.1 Hierarchical Coordinate Assignment

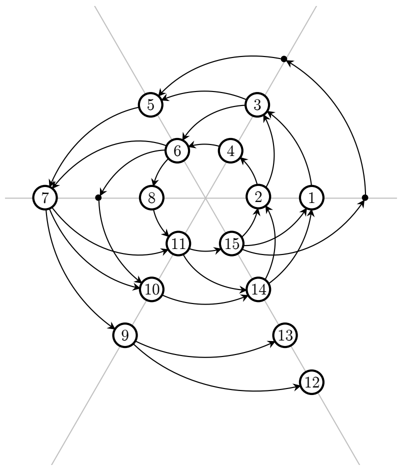
Aligning all vertices leftmost, as in Figure 5.1(a), gives the most narrow drawing possible. However, the result has many bends and no balancing of vertices amongst their neighbors. Several more sophisticated heuristics have been proposed.

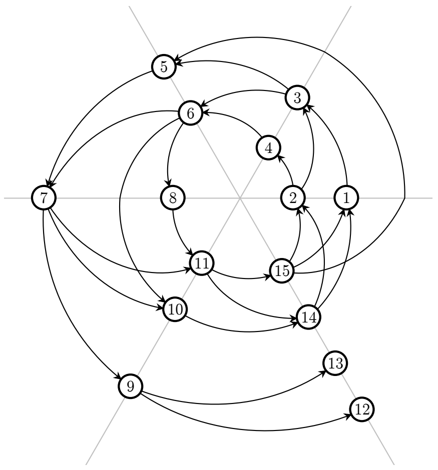
Sugiyama et al. [140] suggest two algorithms for the fourth phase. The first uses a quadratic programming [115] approach. In this approach a parameter $c \in [0, 1]$ determines whether the closeness of incident vertices or balancing a vertex amongst its neighbors is more important. The second heuristic is the priority layout method which uses several up and down sweeps similar to many crossing reduction techniques. In the down sweeps the x-coordinates of the vertices on the current level i are adapted while the x-coordinates on the level $i-1$ are fixed. The heuristic has to ensure that the order of the vertices on level i is not changed. Dummy vertices are given a high priority and moved first to achieve vertically aligned long edges. While moving higher priority vertices, vertices with lower priority are moved with them if necessary. To position vertices with lower priority, heuristics like barycenter [140] and median [51] can be used.

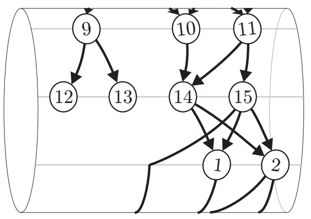
Gansner et al. [66] try to minimize the weighted sum of the difference of x-coordinates for adjacent vertices while keeping at least unit distance for vertices on the same level. Segments between dummy vertices are given the highest weight to align them as vertical as possible. In the same paper the authors suggest to use an auxiliary graph to model the coordinate assignment as a leveling problem.

Eades et al. [48] spread the vertices of the first and last levels equidistantly on their levels and put all other vertices in the barycenter of their neighbors. They use their algorithm only for special graphs in which, e. g., all sources and sinks are on the extreme levels. Note that this approach may change the computed permutations of each level.

The algorithm of Buchheim et al. [21] guarantees at most two bends per edge and draws inner segments vertically.


(a) Ordered cyclic level graph G of G^*


(b) Cyclic 2D drawing of G


(c) Cyclic 3D drawing of G

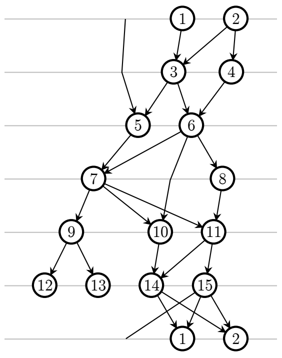

(d) Intermediate drawing of G

Figure 5.2. Cyclic coordinate assignment and postprocessing

5.1.1 Algorithm by Brandes and Köpf

One standard algorithm for the hierarchical coordinate assignment was proposed by Brandes and Köpf [19]. We describe it in more detail here as our cyclic coordinate assignment algorithm is an adaption of their method to cyclic level graphs.

The algorithm guarantees at most two bends per original edge which occur at the topmost and lowest dummy vertex of such an edge. The inner segments are aligned vertically. However, the heuristic tries to align even more segments. In its first phase, all vertices are aligned with their median neighbors if possible by removing all other incident segments. Here, two cases are possible: Align each vertex with the median of its incoming neighbors (align upwards) or align each vertex with the median of the outgoing neighbors (align downwards). If the number of neighbors is even, the median is not unique. Again, two approaches are possible: Choose the left median (align left) or the right median (align right). Hence, the alignment and the subsequent phase is executed four times, i. e., once for each combination of upwards and downwards with left and right alignment. We call each such execution one *run* and describe the upward alignment to the left only.

In the alignment phase, segments of the graph are removed until each vertex has at most one incoming and one outgoing segment and there are no crossings in a drawing respecting the ordering of the vertices on each level. In a type 1 conflict, i. e., the crossing of an outer and an inner segment, the outer segment is removed. As there are no type 2 conflicts, i. e., crossings of inner segments, all inner segments remain. For each vertex, aligning it with its left median incoming neighbor is tried first. If that is not possible, the right median is tried. Each resulting path is called a *block* and drawn vertically. The blocks can be topologically sorted [30] according to the left-to-right order of each level and are compacted by applying a modified longest path leveling then. This gives x-coordinates for each (dummy) vertex for the left upward alignment run.

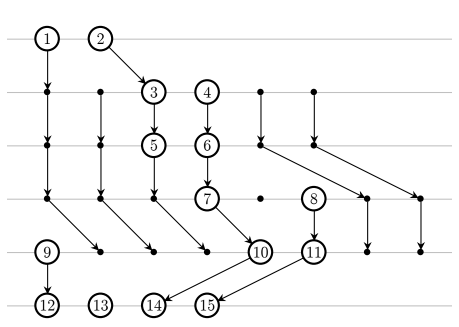
The final step is to balance the x-coordinates of the four runs to one final x-coordinate. This is essentially done by choosing the average of the two median x-coordinates.

Examine the graph in Figure 5.1(a). Removing segments to align vertices upwards to the left results in the graph in Figure 5.3(a). Each path in the graph is a block, which is compacted in Figure 5.3(b). The final drawing, resulting from the balancing step of the four drawings, is given in Figure 5.1(b).

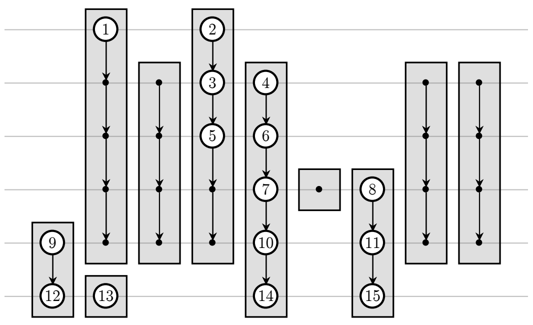
5.1.2 Postprocessing

After the x-coordinates for all (dummy) vertices are computed, an additional straight forward postprocessing is needed.

- All dummy vertices are removed and replaced by edge bends where necessary.
- All edges, which were reversed in the decycling phase, are put in their original direction again.



(a) Remove edges to align vertices



(b) Compact blocks

Figure 5.3. Example of the algorithm of Brandes and Köpf [19] of G^* in Figure 5.1

- The computed x- and y-coordinates are multiplied with user-given values to scale the drawing to the desired size.

If there are vertices with arbitrary size, in some cases, additional bends have to be added to avoid outgoing edges of a vertex v to cross a larger vertex on the same level $\phi(v)$. Sander [128, 129] uses orthogonal edge routing to avoid such crossings. Gansner et al. [66] use Bézier curves [59] to achieve a similar effect without bends.

5.2 Preliminaries

Let $G^* = (V^p, E^p, \phi^p, <)$ be an ordered proper cyclic k-level graph. We denote with $v_j^{(i)} \in V_i$ the j-th vertex on level i in G and $\text{pos}(v)$ the position of the vertex v on its level. We repeat the definition of conflicts for clarity. Two segments in G^p starting at the same level have a *conflict* if they cross or share a vertex. Conflicts are of *type 0, 1* or *2*, if none, one or two of the conflicting segments are inner segments, respectively. Note that type 1 or type 2 conflicts can only occur if the two segments cross. Type 0 conflicts can be due to a crossing of two outer segments or the two segments have a common start or end vertex.

5.3 Cyclic Coordinate Assignment

In this section, we describe our coordinate assignment phase for cyclic level graphs. We adapt the algorithm of Brandes and Köpf [19] and use their notation. Like them, we also assume that the crossing reduction has avoided type 2 conflicts. These can be avoided, e. g., by using our global sifting crossing reduction presented in Chapter 4.4.

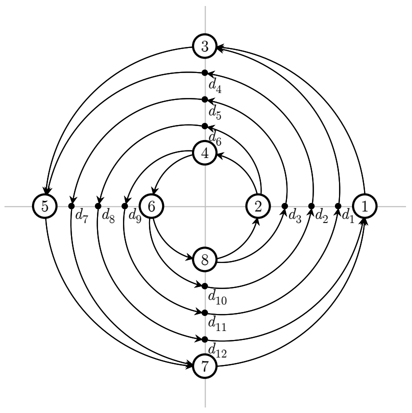
The input to our algorithm is the output of the third phase, i. e., a proper ordered cyclic level graph. Note that dummy vertices were introduced after the leveling.

Algorithm 5.1. cyclicCoordinateAssignment

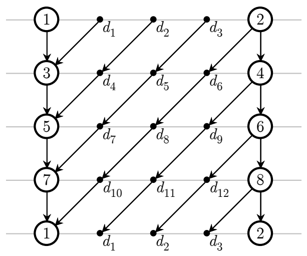
```
Input: $G^p = (V^p, E^p, \phi^p, <)$: An ordered and proper cyclic $k$-level graph
Output: Coordinates $(x(v), y(v))$ for each $v \in V^p$ in the intermediate
        drawing $\mathcal{I}$

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

Algorithm 5.1 consists of three basic steps: block building (lines 4–5), horizontal compaction (lines 6–12), and balancing (line 14) which correspond to the steps in [19]. 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 where necessary.

In the cyclic case, there may be unavoidable cyclic dependencies in the left-to-right ordering among vertically aligned paths which are the main new challenge in the cyclic coordinate assignment. Consider the graph shown in Figure 5.4 to Figure 5.6 for an example. In each figure, the left hand side shows a cyclic drawing of the same graph. The right hand side shows the corresponding intermediate drawing. If we were to draw all inner segments vertically this would lead to the cyclic dependency $x(d_1) < x(d_2) < x(d_3) = x(d_5) = x(d_7) < x(d_8) < x(d_9) = x(d_{11}) = x(d_1)$, which is a contradiction. We will later formalize these cyclic dependencies and call them rings. In the intermediate drawing (see Figure 5.5(b)), the vertices on the two copies of the first level do not have the same x-coordinate then. The two copies of these vertices cannot be drawn on the same point in the cyclic drawing (see Figure 5.5(a)). One possibility to solve this problem is to allow additional bends, e. g., between the last and the first level. Then the two copies of each vertex on level 1 have the

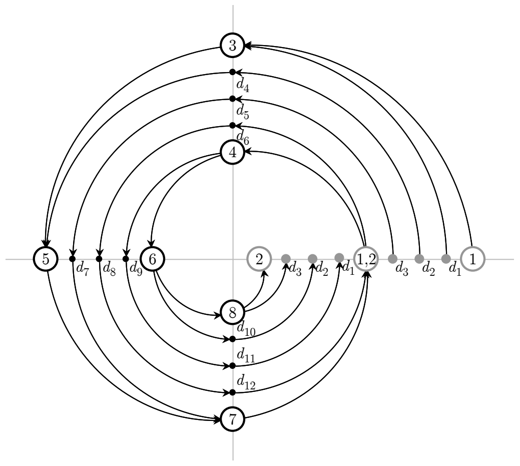


(a) Cyclic drawing

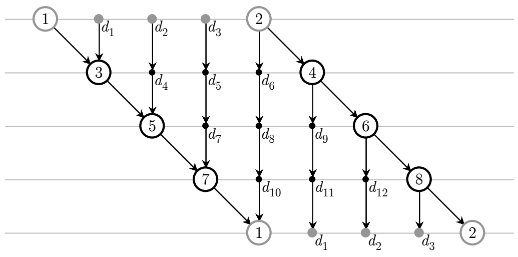


(b) Intermediate drawing

Figure 5.4. Example graph with sheared inner segments

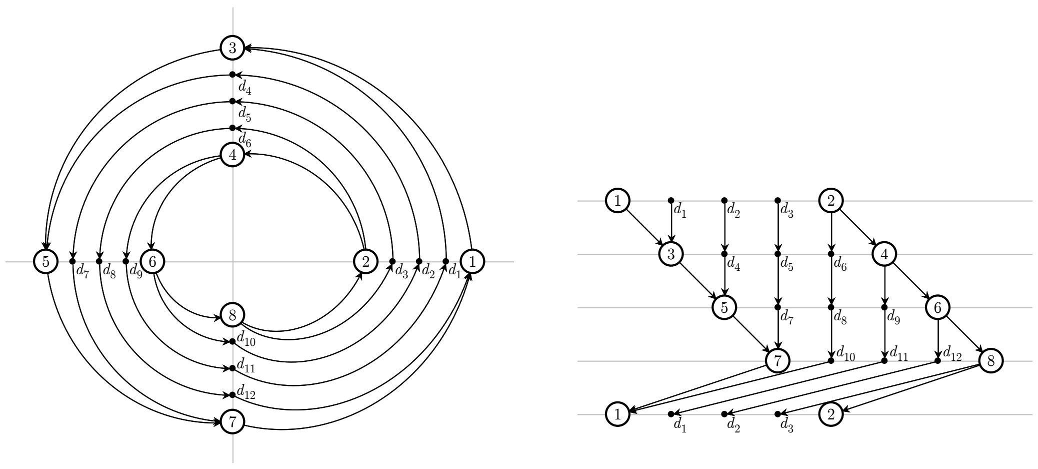


(a) Cyclic drawing



(b) Intermediate drawing

Figure 5.5. Example graph with vertical inner segments



(a) Cyclic drawing (b) Intermediate drawing

Figure 5.6. Example graph with additional bends

same x-coordinate (see Figure 5.6(b)) and the cyclic drawing is at least possible (see Figure 5.6(a)). However, edges can then have up to four bends. But one main goal of the cyclic framework is to avoid any special treatment of edges from the last to the first level, since such an approach would destroy the symmetry of the graph completely. Another possibility is to align inner vertices using the same slope (see Figure 5.4(a) and Figure 5.4(b)). Then the symmetry of a graph can still be represented with at most two bends per edge. We follow this approach in our algorithm. The main new problems that arise herewith are to compute the slope and to determine how many different slopes are needed.

Such cyclic dependencies can be avoided completely by using our global sifting heuristic for the crossing reduction (see Chapter 4.4) and a slight modification of our coordinate assignment algorithm in this chapter. We will give details in Section 5.5. However, it is important to have an algorithm capable of treating these dependencies nevertheless. Note that the graph in Figure 5.4 can only be drawn without crossings with the permutations of each level given in Figure 5.4 (or all permutations reversed). Hence, these permutations could be the result of an optimal crossing reduction or a planarity testing and embedding algorithm.

5.3.1 Block Building

For building the blocks, the same approach as in the algorithm by Brandes and Köpf [19] is used. Please note that the blocks used for the coordinate assignment here and in [19] differ slightly from the blocks used in Chapter 4 for our global sifting algorithm. There, all inner segments of each edge build one block. In the coordinate assignment, all inner segments of each edge are in the same block as well. However, the block can be larger and contain additional inner and outer segments.

Nevertheless, we use a very similar notation.

We try to align each vertex with the median of all its upper adjacent vertices by assigning them to the same block. Removing all other segments not used for aligning results in a graph in which each vertex has at most one incoming and one outgoing segment. We call such a graph a cyclic path graph.

Definition 5.1. *A* cyclic path graph $H' = (V, E_{\text{intra}}, \phi, <)$ *is a plane ordered proper cyclic level graph. Each vertex of H' has in-degree and out-degree at most one. We call each connected component of H' a* block *and all edges $e \in E_{\text{intra}}$, i. e., all edges of H',* intra block edges. *A block B is* closed *if each vertex of B has in-degree and out-degree one and B is* open, *otherwise. The* height *of B is defined as the number of intra block edges in B. Let* $\text{block}(v)$ *be the block of a vertex $v \in V$. For an open block B, let* $\text{top}(B)$ *and* $\text{bottom}(B)$ *be the topmost and lowest vertex, i. e., the vertices with in-degree and out-degree zero, respectively. Let $\mathcal{B}$ be the set of all blocks and* $\text{levels}(B)$ *the set of levels on which B has (dummy) vertices.*

The cyclic block graph $H = (V, E_{\text{intra}} \dot\cup E_{\text{inter}}, \phi)$ *of H' is obtained by adding an edge $e \in E_{\text{inter}}$ from each vertex in H' to its consecutive right vertex on the same level (if there is one), which we call* inter block edges E_{inter}. *For a vertex $v \in V$ let* $\text{left}(v)$, $\text{right}(v)$, $\text{up}(v)$, *and* $\text{down}(v)$ *be the start vertex of the incoming inter block edge, the end vertex of the outgoing inter block edge, the start vertex of the incoming intra block edge, and the end vertex of the outgoing intra block edge (if they exist), respectively.*

A hierarchical block graph *is defined analogously being an ordered proper (non-cyclic) level graph.*

See Algorithm 5.2 for the creation of the cyclic block graph. We traverse each level separately (line 3). We first mark outer segments involved in type 1 conflicts between the current level j and the level i above. To do so, we traverse the lower level (lines 7–18) and stop at each end vertex of an inner segment and after the last vertex on the level. We traverse all (non-dummy) vertices l between the last and the current inner segment on level j (lines 13–17). If an incoming segment of the (non-dummy) vertex l starts left of the last found inner segment ($k < k_0$) or starts right of the current inner segment ($k > k_1$), it is marked as it crosses at least one inner segment. See Figure 5.7 for an illustration of this situation where marked segments are dashed.

Then, we traverse the current level j from left to right again and try to align each vertex $v_c^{(j)}$ with one of its median predecessor vertices (lines 20–28). First we try

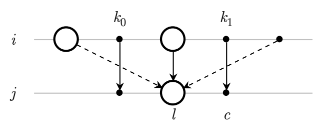


Figure 5.7. Marking type 1 conflicts

Algorithm 5.2. buildCyclicBlockGraph

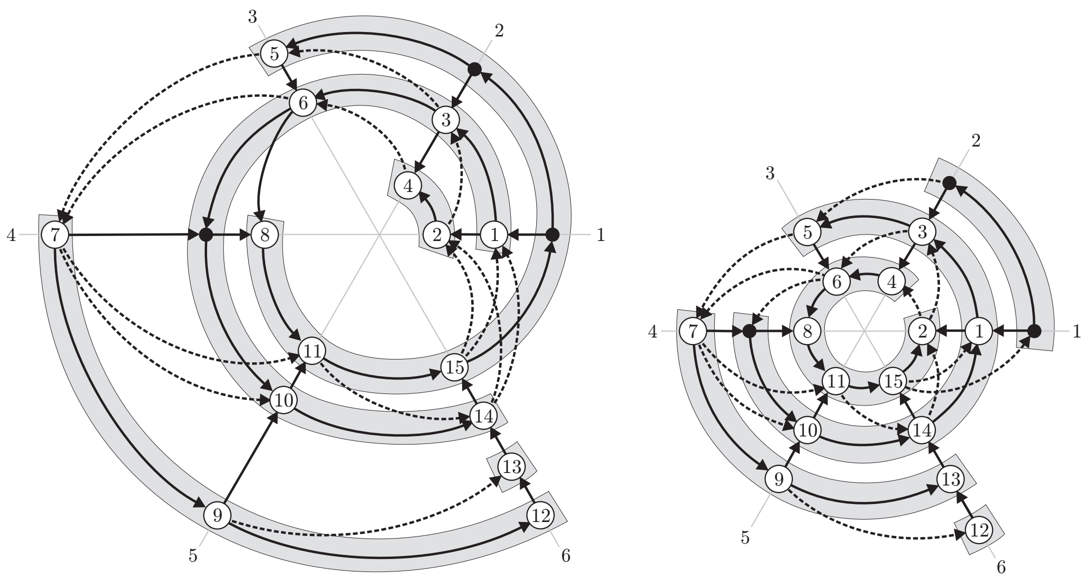
```
Input: $G^p = (V^p, E^p, \phi^p, <)$: An ordered and proper cyclic $k$-level graph
Output: The cyclic block graph $H$ of $G^p$

1  $E_{\text{intra}} \leftarrow \emptyset$
2  $E_{\text{inter}} \leftarrow \emptyset$
3  for $1 \leq i \leq k$ do
4      $j \leftarrow \text{next}(i)$
5      $k_0 \leftarrow 0$
6      $l \leftarrow 1$
7      for $1 \leq c \leq |V_j^p|$ do
8          if $c = |V_j^p|$ or $v_c^{(j)}$ end vertex of inner segment then
9              if $v_c^{(j)}$ end vertex of inner segment $s$ then
10                 $k_1 \leftarrow$ position of start vertex of $s$ in $V_i^p$
11             else
12                 $k_1 \leftarrow |V_i^p|$
13             while $l \leq c$ do
14                 foreach upper neighbor $v_k^{(i)}$ of $v_l^{(j)}$ do
15                     if $k < k_0$ or $k > k_1$ then
16                         mark segment $(v_k^{(i)}, v_l^{(j)})$
17                 $l \leftarrow l + 1$
18             $k_0 \leftarrow k_1$
19     $r \leftarrow 0$
20     for $1 \leq c \leq |V_j^p|$ do
21         if $c < |V_j^p|$ then
22             $E_{\text{inter}} \leftarrow E_{\text{inter}} \cup \{(v_c^{(j)}, v_{c+1}^{(j)})\}$
23         if $v_c^{(j)}$ has upper neighbors $u_1 < \ldots < u_d$ with $d > 0$ then
24             for $\lfloor \frac{d+1}{2} \rfloor \leq m \leq \lceil \frac{d+1}{2} \rceil$ do
25                 if $\text{up}(v_c^{(j)}) = \text{null}$ then
26                     if $(u_m, v_c^{(j)})$ not marked and $r < \text{pos}(u_m)$ then
27                         $E_{\text{intra}} \leftarrow E_{\text{intra}} \cup \{(u_m, v_c^{(j)})\}$
28                         $r \leftarrow \text{pos}(u_m)$
29 $H \leftarrow (V^p, E_{\text{intra}} \dot\cup E_{\text{inter}}, \phi^p)$
30 return $H$
```

Algorithm 5.3. splitLongBlocks

```
Input: $H = (V, E_{\text{intra}} \dot\cup E_{\text{inter}}, \phi)$: A cyclic $k$-level block graph
Output: The cyclic block graph $H$ with all blocks having height $\leq k-1$
for $B \in \mathcal{B}$ do
    if $B$ is a closed block then
        remove arbitrary outer segment of $B$ from $E_{\text{intra}}$
    else if height$(B) \geq k$ then
        $v \leftarrow$ top$(B)$
        $v_0 \leftarrow \emptyset$
        $w \leftarrow 0$
        $w_0 \leftarrow 0$
        while down$(v) \neq$ null do
            $w \leftarrow w + 1$
            if $(v, \text{down}(v))$ inner segment then
                $v_0 \leftarrow v$
                $w_0 \leftarrow w$
            if $w = k$ then
                $E_{\text{intra}} \leftarrow E_{\text{intra}} \setminus \{(v_0, \text{down}(v_0))\}$
                $w \leftarrow w - w_0$
            $v \leftarrow$ down$(v)$
return $H$
```

its upper left median, then its upper right median (lines 24–28). The second test is skipped if the first was successful (line 25). Using a segment for aligning is impossible if the segment is marked or if it would cross a segment already used for aligning. The current vertex $v_c^{(j)}$ becomes the top vertex of a new block if both alignments fail or if the vertex does not have any upper neighbors. As outer segments involved in type 1 conflicts are never used for aligning and as there are no type 2 conflicts, all inner segments of the graph are used for aligning. Hence, all inner segments of an edge are in the same block. As each block is drawn with constant slope, this ensures at most two bends per edge. All segments not used for aligning are removed to build the cyclic path graph. E_{intra} is the set of all remaining segments. See Figure 5.8 for examples of cyclic block graphs of G^*. Vertices and intra block edges of the same block are framed. The dotted segments were removed in the block building phase. The cyclic block graph contains the additional inter block edges which lie on the level lines pointing to the center. Note that a drawing of the cyclic block graph, respecting the ordering of the vertices on each level, does not have any crossing.

The cyclic block graph can have closed blocks (with height k) and open blocks with height $\geq k$ (spirals) which we avoid to simplify our coordinate assignment algorithm (see Algorithm 5.3). In a closed block, we remove an arbitrary outer



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

Figure 5.8. Block graphs of Figure 5.2(a) as cyclic 2D drawings

segment to get an open block with height $k-1$ (lines 2–3). In a spiral, we traverse the block from the topmost to the lowest segment and iteratively remove the latest possible outer segment such that the block above has at most height $k-1$ (lines 4–17). In both cases, 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 is not sheared like other blocks in Section 5.3.2. It cannot be part of a cyclic dependency as is splits the graph in two disjoint parts. An open block with height $\geq k$ would have to be sheared anyway as it alone contains a cyclic dependency. See Figure 5.8(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 $k-1$.

5.3.2 Horizontal Compaction

In this section, we describe how to compact the cyclic block graph by arranging all blocks as close to each other as possible and, hence, 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 5.2. *A* block path *P in a cyclic block graph $H = (V, E_{\text{intra}} \dot\cup E_{\text{inter}}, \phi)$ is a sequence of vertices $v_1, \ldots, v_s \in V$ such that for each pair of consecutive vertices v_i and $v_{i+1}, 1 \leq i < s$, $(v_i, v_{i+1}) \in E_{\text{intra}}$ or $(v_{i+1}, v_i) \in E_{\text{intra}}$ or $(v_i, v_{i+1}) \in E_{\text{inter}}$. It* *is* simple *if all vertices are mutually distinct. A block path is a* ring R *if* $v_1 = v_s$ *and if it traverses at least one inter block edge. 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 defined as* $\text{wind}(R) = (c_{down} - c_{up})/k$*.*

Informally, a ring is a circle in the block graph where the direction of the inter block edges is preserved and the intra block edges are used in any direction. Survey Figure 5.5(b) and the cyclic dependency $x(d_1) < x(d_2) < x(d_3) = x(d_5) = x(d_7) < x(d_8) < x(d_9) = x(d_{11}) = x(d_1)$ again: In the cyclic block graph of this graph the sequence of vertices $d_1, d_2, d_3, d_5, d_7, d_8, d_9, d_{11}, d_1$ is a ring. Each inequality represents an inter block edge traversed from left to right and each equality represents an intra block edge traversed upwards or downwards. The width of this ring is 4 as four inter block edges are used. This corresponds to the fact that in the drawing in Figure 5.5(b) the lower copy of level 1 is four units shifted to the right compared to the upper copy of level 1. Without the additional condition of a ring using at least one inter block edge, the sequence d_3, d_5, d_7, d_5, d_3 would be a ring as well. However, such a sequence does not pose a problem, as we never leave the block and can assign all vertices of the sequence the same x-coordinate. As we avoid spirals, i. e., blocks with height $\geq k$, the start and end vertex of each inter block edge lie in different blocks and, hence, each ring traverses at least two blocks.

$\text{wind}(R)$ counts how often a ring 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 treat the strongly connected components (SCCs) connected by block paths of the block graph separately. There are two types of SCCs: The *simple SCCs* consist of one block. All other *complex SCCs* contain rings. Figure 5.8(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 $\text{wind}(R) = 1$. We now prove several properties of rings.

Lemma 5.1. *A hierarchical block graph* G *cannot contain a ring.*

Proof. We prove this by constructing real x-coordinates for each vertex such that all vertices in the same block have the same x-coordinate. We assign the j-th vertex on level 1 the x-coordinate j. When treating a level $i > 1$, we first assign each vertex v on level i, which is in the same block as a vertex u on level $i - 1$, the x-coordinate $x(v) = x(u)$. This does not contradict the ordering of the level i as there are no type 2 conflicts. Let $v_1, \ldots, v_l$ be all vertices on level i which have x-coordinates now. For each $1 \leq m < l$, we assign all vertices between v_m and v_{m+1} increasing x-coordinates between $x(v_m)$ and $x(v_{m+1})$ respecting the permutation of level i. We assign all vertices left of v_1 and all right of v_l x-coordinates respecting the permutation smaller than $x(v_1)$ and larger than $x(v_l)$, respectively.

This procedure constructs x-coordinates for each vertex, with all vertices in the same block having the same x-coordinate. As this is not possible if G contains a ring, this proves the statement. $\square$

Lemma 5.2. *For each ring R of a cyclic block graph of a ordered proper cyclic level graph G,* $\operatorname{wind}(R) \neq 0$.

Proof. Assume for contradiction that there exists a ring R with $\operatorname{wind}(R) = 0$ in the cyclic 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, eventually leads to a block graph H' of a (non-cyclic) level graph G', which contains R completely. This is a contradiction according to Lemma 5.1. □

Lemma 5.3. *For each simple ring R of a cyclic block graph,* $|\operatorname{wind}(R)| \leq 1$.

Proof. Assume for contradiction that there is a simple ring R with $|\operatorname{wind}(R)| > 1$. As R wraps around the center in the 2D drawing more than once, in each drawing of R the corresponding curve crosses itself. As R is simple the crossing cannot be due to a common vertex.

Each cyclic path graph is plane. Further, each drawing of it respecting its ordering can be extended to a plane 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. □

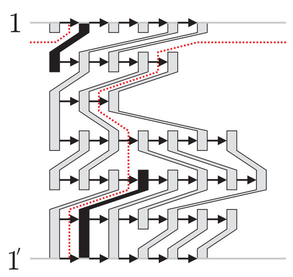
Theorem 5.1. *Let $\mathcal{R}$ be the set of all simple rings of an SCC in a cyclic block graph. Either for each $R \in \mathcal{R}$* $\operatorname{wind}(R) = 1$ *or for each $R \in \mathcal{R}$* $\operatorname{wind}(R) = -1$.

Proof. According to Lemmas 5.2 and 5.3, $|\operatorname{wind}(R)| = 1$ holds for each ring $R \in \mathcal{R}$. Assume for contradiction that there exist two rings $R_1, R_2 \in \mathcal{R}$ with $\operatorname{wind}(R_1) = 1$ and $\operatorname{wind}(R_2) = -1$. Let v_1 and v_2 be vertices in R_1 and R_2, respectively, lying in different blocks. Such vertices exist, as each ring traverses at least two blocks. Let P_1 be a block path from v_1 to v_2 and P_2 be a block path from v_2 to v_1. Both exist due to the strong connectivity. Concatenating P_1 and P_2 results in a (not necessarily simple) ring S through v_1 and v_2. Due to Lemma 5.2, $\operatorname{wind}(S) \neq 0$. If $\operatorname{wind}(S) > 0$, let T be a non-simple ring consisting of S and $\operatorname{wind}(S)$ many copies of R_2 joined via v_2. Otherwise, let T be a ring consisting of S and $-\operatorname{wind}(S)$ many copies of R_1 joined via v_1. In both cases $\operatorname{wind}(T) = 0$, which contradicts Lemma 5.2. □

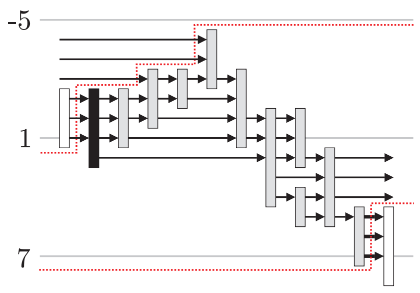
Definition 5.3. *Let S be a complex SCC of a cyclic block graph containing a simple ring R. We define* $\operatorname{wind}(S) = \operatorname{wind}(R)$*, which is well-defined according to Theorem 5.1, and* $\operatorname{width}(S)$ *as the maximum width of all simple rings in S.*

5.3.2.1 Compaction of a Complex Strongly Connected Component

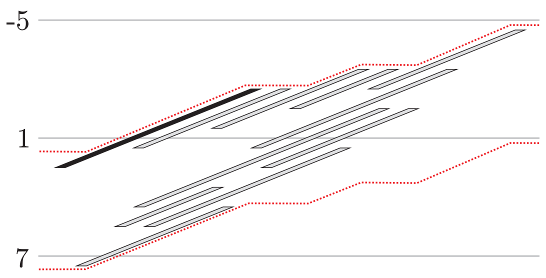
It is not possible to draw all blocks of a complex SCC S straight-line and vertically. Therefore, we draw all blocks of S with equal slope. The slope has to be chosen such that each ring and, hence, the resulting curve in S starts and ends at the same coordinate. All rings in S have the same number of windings $\operatorname{wind}(S)$, which is either 1 or -1. Therefore, each simple ring spans $\operatorname{wind}(S) \cdot k$ levels. To draw one simple ring R, we could use the slope $-(\operatorname{wind}(R) \cdot k)/\operatorname{width}(R)$, which would result



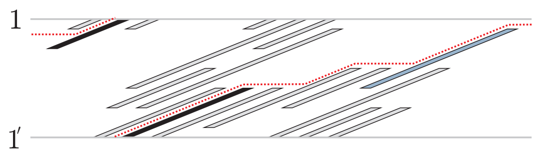
(a) Left-aligned intermediate drawing



(b) Compacted drawing



(c) Sheared drawing



(d) Final intermediate drawing

Figure 5.9. Drawing of a complex SCC

in inter block edges of unit length. In order to draw all blocks in S with the same slope (line 10 in Algorithm 5.1), we must use the width of the widest simple ring of S and the slope $-(\text{wind}(S)\cdot k)/\,\text{width}(S)$. With this slope, the widest ring fits exactly and uses unit length inter block edges. All narrower rings have some unused horizontal space in the drawing and have inter block edges which are longer than one unit. Examine Figure 5.4 again. The inner segments build a SCC S for which $\text{width}(S) = 4$ and $\text{wind}(S) = 1$ holds and the graph uses $k = 4$ levels. Hence, the slope of $-(\text{wind}(S)\cdot k)/\,\text{width}(S) = -(1\cdot 4)/4 = -1$ is used for these inner segments in Figure 5.4(b). Note that the positive y-axis points downwards in intermediate drawings.

To use the slope $-(\text{wind}(S)\cdot k)/\,\text{width}(S)$, we have to determine $\text{wind}(S)$ and $\text{width}(S)$. $\text{wind}(S)$ is determined by $\text{wind}(R)$ for an arbitrary ring R in S. Hence, searching for an simple ring R in S suffices, which can easily be done in linear time. We assume $\text{wind}(S) = 1$ in the following paragraphs. Computing $\text{width}(S)$ corresponds to determining the width of the widest simple ring in S. In general, the problem of finding the longest cycle in a directed graph is $\mathcal{NP}$-hard [69]. But complex SCCs are very special graphs, which can be drawn without crossings with the given leveling. Finding the widest simple ring can be solved in linear time by cutting the SCC. The idea is to cut along a special block path in the cyclic block graph from a leftmost vertex on its level to a rightmost vertex on its level. Finding the width of the SCC and compacting the layout is done simultaneously then.

5.3.2.1.1 Cutting a Strongly Connected Component

Lemma 5.4. *Let S be an SCC with* $\mathrm{wind}(S) = 1$. *Then, there exists a block B such that its lowest vertex is leftmost on its level.*

Proof. Assume for contradiction that each block has a lowest vertex with a left neighbor and, hence, an incoming inter block edge. Starting at an arbitrary vertex, we traverse downwards in intra block direction to the lowest vertex of its block and then against inter block direction to a new block. We repeat this until an already traversed vertex v is found. Reversing the sequence from v to v results in a circle using all inter block edges in their direction and, consequently, in a ring R. As all intra block edges are used against their direction now, $\mathrm{wind}(R) = -1$ holds. This is a contradiction to $\mathrm{wind}(S) = 1$ according to Theorem 5.1. □

W. l. o. g. let $\mathrm{wind}(S) = 1$. To cut the SCC S (see Algorithm 5.4), we build a path P starting at a leftmost vertex v on its level which is the lowest vertex in its block B (line 1). Such a vertex and block always exist due to Lemma 5.4. We traverse the block to the topmost vertex and use the outgoing inter block edge of that vertex to reach a new block. We repeat this procedure until we reach a block with topmost vertex which does not have an outgoing inter block edge and is, hence, a rightmost vertex on its level (lines 4–12). This has to happen eventually as otherwise a vertex u would have to be visited a second time. But then the sequence between the two occurrences of u is a ring R with $\mathrm{wind}(R) = -1$ which contradicts $\mathrm{wind}(S) = 1$. Hence, P is a path from a leftmost vertex to a rightmost vertex.

Let P consist of $v_1, \dots, v_m$. For each vertex v_i $(i > 1)$, we remove its incoming inter block edge if it does not start at v_{i-1} (lines 9–10). In other words, we remove all incoming inter block edges not belonging to P and, thus, all those edges left of P. Note that P is only constructed for clarity in Algorithm 5.4 and not needed, as we cut the inter block edges while traversing from left to right.

We construct a hierarchical block graph S' corresponding to the cyclic block graph in the following way: We assign the vertex $v \in V$ the new level $\phi'(v) = \phi(v)$ (line 15). In a traversal of the block graph, we assign each vertex a new level ϕ': Using an inter block edge (in any direction), we assign both end vertices the same level ϕ' (lines 19–24). Using an intra block edge in or against its direction, we increase (lines 28–30) or decrease (lines 25–27) the level ϕ' by 1, respectively, without using a modulo operation.

See Figure 5.9(a) for an example. The cutting of the SCC starts at the bottom of the black block and the dashed path crosses all removed inter block edges. Figure 5.9(b) shows the resulting hierarchical block graph, where we allow negative levels exceptionally.

The removed incoming inter block edges left of P cut all simple rings in S exactly once as the following lemma shows.

Algorithm 5.4. cutSCC

```
Input: $S = (V, E_{\text{intra}} \dot\cup E_{\text{inter}}, \phi)$: A SCC of a cyclic $k$-level block graph
Output: A hierarchical block graph $S'$, set of removed inter block edges $R$
$v \leftarrow$ arbitrary leftmost vertex on its level with $v = \text{bottom}(\text{block}(v))$
$P \leftarrow \emptyset$
$R \leftarrow \emptyset$
while $v \neq$ null do
    $P \leftarrow P \circ (v)$
    if $\text{up}(v) \neq$ null then
        $v \leftarrow \text{up}(v)$
        if $\text{left}(v) \neq$ null then
            $E_{\text{inter}} \leftarrow E_{\text{inter}} \setminus \{(\text{left}(v), v)\}$
            $R \leftarrow R \cup \{(\text{left}(v), v)\}$
    else
        $v \leftarrow \text{right}(v)$
for $v \in V$ do
    $\phi'(v) \leftarrow$ null
$\phi'(v) \leftarrow \phi(v)$
$Q \leftarrow \{v\}$
while $Q \neq \emptyset$ do
    $u \leftarrow Q.remove$
    if $\text{left}(u) \neq$ null and $\phi'(\text{left}(u)) =$ null then
        $\phi'(\text{left}(u)) \leftarrow \phi'(u)$
        $Q.add(\text{left}(u))$
    if $\text{right}(u) \neq$ null and $\phi'(\text{right}(u)) =$ null then
        $\phi'(\text{right}(u)) \leftarrow \phi'(u)$
        $Q.add(\text{right}(u))$
    if $\text{up}(u) \neq$ null and $\phi'(\text{up}(u)) =$ null then
        $\phi'(\text{up}(u)) \leftarrow \phi'(u) - 1$
        $Q.add(\text{up}(u))$
    if $\text{down}(u) \neq$ null and $\phi'(\text{down}(u)) =$ null then
        $\phi'(\text{down}(u)) \leftarrow \phi'(u) + 1$
        $Q.add(\text{down}(u))$
$S' \leftarrow (V, E_{\text{intra}} \dot\cup E_{\text{inter}}, \phi')$
return $(S', R)$
```

Lemma 5.5. *Let S be a complex SCC of a cyclic block graph with* $\operatorname{wind}(S) = 1$. *Removing inter block edges as described removes exactly one inter block edge from each simple ring in S.*

Proof. Removing the inter block edges left of P results in the hierarchical block graph S' which does not contain any ring due to Lemma 5.1. Hence, each ring is cut at least once.

Let R be a simple ring in S. Assume for contradiction that $b > 1$ inter block edges of R were removed. Each such edge corresponds to jumping from the bottom end of S' to the top end of S'. But then $\operatorname{wind}(R) = b > 1$ which is a contradiction to Lemma 5.3. □

Note that it is not possible for a ring to cross P the other way round, i. e., to jump from the top end to the bottom end as this would mean to traverse a inter block edge backwards.

5.3.2.1.2 Actual Compacting We compact the hierarchical block graph S' (in contrast to [19]) in the following way (see Algorithm 5.5): We maintain an array X which stores at $X[i]$ the last used x-coordinate on level i (lines 2–6). 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 [30], and move them as much to the left as possible, preserving unit distance (lines 7–11). 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 (lines 12–18). For placing a block as close as possible to the already placed ones, we traverse its levels (lines 8 and 15). After the compaction, each block (and therefore each vertex v in S') has an assigned x'-coordinate.

5.3.2.1.3 Determining the Slope 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 $\operatorname{width}(S)$ in S (line 19 of Algorithm 5.5).

See Figure 5.9 for an example of an SCC S with $\operatorname{wind}(S) = 1$ and $k = 6$ levels again. The dashed line Figure 5.9(a) cuts six inter block edges. Figure 5.9(b) shows the resulting compacted hierarchical block graph using the leveling ϕ'. The widths of the widest rings through each of the six cut edges are (from top to bottom): 5, 5, 7, 10, 10, 10. Consequently, $\operatorname{width}(S) = 10$. Shearing the drawing with slope $\frac{-1 \cdot 6}{10}$ results in Figure 5.9(c). Using the modulo operation for the y-coordinates gives the final intermediate drawing in Figure 5.9(d). See Algorithm 5.6 for the computation of the coordinates of the intermediate drawing.

Theorem 5.2. *The intermediate drawing of S uses the coordinates* $x(v) = x'(v) - (\operatorname{width}(S)/(\operatorname{wind}(S) \cdot k)) \cdot \phi'(v)$ *and* $y(v) = ((\phi'(v) - 1) \bmod k) + 1 = \phi(v)$ *for each vertex v in S. In the drawing all intra block edges have* $\operatorname{slope}(S) = -(\operatorname{wind}(S) \cdot k)/\operatorname{width}(S)$. *The ordering of vertices on the same level is the ordering given by the crossing reduction phase and these vertices have at least unit distance.*

Algorithm 5.5. compactBlocks

```
Input: $S = (V, E_{\text{intra}} \dot\cup E_{\text{inter}}, \phi')$: A hierarchical $k'$-level block graph and the
       set of removed inter block edges $R$
Output: The width of $S$ and the values $x'(v)$ set for each $v \in V$
Let $\mathcal{B}$ be the list of blocks of $S$ in topological order
$l_{\min} \leftarrow \min_{v \in V}\{\phi'(v)\}$
$l_{\max} \leftarrow \max_{v \in V}\{\phi'(v)\}$
Let $X$ be an array of size $k'$ with indices $[l_{\min}, \dots, l_{\max}]$
for $l_{\min} \le i \le l_{\max}$ do
    $X[i] \leftarrow -1$
for $B \in \mathcal{B}$ do in topological order
    $x \leftarrow \max_{l \in \text{levels}(B)} X[l]$
    foreach $v \in V(B)$ do
        $x'(v) \leftarrow x + 1$
        $X[\phi'(v)] \leftarrow x + 1$
for $B \in \mathcal{B}$ do against topological order
    if $B$ has outgoing inter block edges then
        Let $L$ be the set of levels of $B$ with outgoing inter block edges
        $x \leftarrow \min_{l \in L} X[l]$
        foreach $v \in V(B)$ do
            $x'(v) \leftarrow x - 1$
            $X[\phi'(v)] \leftarrow x - 1$
$w \leftarrow \max_{e=(u,v) \in R} (x'(u) - x'(v) + 1)$
return $w$
```

Algorithm 5.6. shear

```
Input: $S = (V, E_{\text{intra}} \dot\cup E_{\text{inter}}, \phi')$: A hierarchical block graph, slope $s$
Output: A sheared cyclic block graph with $x(v)$ and $y(v)$ set for each $v \in V$
for $v \in V$ do
    $x(v) = x'(v) + \frac{y'(v)}{s}$
    $y(v) = ((\phi'(v) - 1) \mod k) + 1$        // $= \phi(v)$
$S' \leftarrow (V, E_{\text{intra}} \dot\cup E_{\text{inter}}, y)$
return $S'$
```

Proof. In the compacted drawing of the hierarchical block graph, all blocks are drawn vertically. Shearing the drawing results in unchanged ϕ'-coordinates and new x-coordinates $x(v) = x'(v) + \phi'(v)/\operatorname{slope}(S)$. Now all edges have slope $\operatorname{slope}(S)$. Using the y-coordinates $y(v) = ((\phi'(v) - 1) \mod k) + 1 = \phi(v)$ does not affect the slope of the edges but results in the same y-coordinates of all vertices on the same level 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 ϕ'-coordinate 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 $\phi'(v) = \phi'(u) - k \cdot \operatorname{wind}(S)$. 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 $\operatorname{width}(S)$ wide and, thus, $x'(v) \geq x'(u) - (\operatorname{width}(S) - 1)$. 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, hence, all vertices on the same level are separated by at least unit distance and are in the ordering given by the crossing reduction phase. □

5.3.2.2 Compaction of all Compacted Strongly Connected Components

Our next step is to globally compact the set of compacted complex SCCs and simple SCCs.

Lemma 5.6. *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 such that 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 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 like we compact the blocks of a hierarchical block graph. One slight complication is that the SCCs do not have a smooth left and right border. To compute the shift of one SCC we simply traverse all vertices of the SCC. Algorithm 5.7 shows the details of this compaction step.

5.3.3 Balancing

In this phase (see Algorithm 5.8), the four results are balanced by computing one x-coordinate for each vertex from the four x-coordinates computed by the four preceding runs. The only difference to the algorithm of Brandes and Köpf [19] is that we do not use the average median of the four x-coordinates for each vertex since

Algorithm 5.7. compact

```
Input: $\mathcal{S}$: Set of compacted SCCs of a cyclic $k$-level graph
Output: The values $x(v)$ set for each vertex $v$ in each SCC of $\mathcal{S}$
1  Let $\mathcal{S}$ be the list of SCCs in topological order
2  Let $X$ be an array of size $k$ with indices $[1, \ldots, k]$
3  for $1 \leq i \leq k$ do
4      $X[i] \leftarrow -1$
5  for $S \in \mathcal{S}$ do in topological order
6      $\delta_x \leftarrow \min_{v \in V(S)}(x'(v) - X[\phi(v)])$
7      foreach $v \in V(S)$ do
8          $x(v) \leftarrow x'(v) - \delta_x + 1$
9          if $x(v) > X[\phi(v)]$ then
10             $X[\phi(v)] \leftarrow x(v)$
11 for $S \in \mathcal{S}$ do against topological order
12     if $S$ has outgoing inter block edges into another SCC then
13         Let $M$ be the set of vertices of $S$ with outgoing inter block edges into
           another SCC
14         $\delta_x \leftarrow \min_{v \in M}(X[\phi(v)] - x(v))$
15         foreach $v \in V(S)$ do
16             $x(v) \leftarrow x(v) + \delta_x - 1$
17             if $x(v) < X[\phi(v)]$ then
18                 $X[\phi(v)] \leftarrow x(v)$
```

Algorithm 5.8. balance

```
Input: $\mathcal{P}$: Set of the results of the four runs of $G^p = (V^p, E^p, \phi^p, <)$
Output: $x(v)$ and $y(v)$ for each vertex $v \in V^p$
1  Let $x_i(v)$ be the $x$-coordinate of the vertex $v$ in the $i$-th run $(1 \leq i \leq 4)$
2  foreach $v \in V^p$ do
3      $x(v) \leftarrow (x_1(v) + x_2(v) + x_3(v) + x_4(v))/4$
4      $y(v) \leftarrow \phi^p(v)$
```

this can induce additional bends on edges 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. The result is a non-integer x-coordinate. However, due to the sheared drawing of the SCCs, none of the four intermediate drawings has guaranteed integer x-coordinates, anyway.

Proposition 5.1. *Using the average of the x-coordinates of the four runs to determine the final x-coordinate for each vertex does not change the ordering of the vertices on a level and preserves at least unit distance. It does not add bends to subgraphs, which belong to an SCC in all four runs. 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. Consequently, balancing can lead to more different slopes than in each of the four runs alone. See Figure 5.8 for two different block graphs of two runs of the same graph.

If similar slopes are more important than balancing, we only compute one downwards run with a modified block building phase. 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 traverse 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.

5.4 Algorithm Analysis

Theorem 5.3. *Let $G = (V, E, \phi, <)$ be a (not necessarily proper) ordered cyclic k-level graph. The width of the intermediate drawing of G is $\mathcal{O}((|V| + |E|)^2)$ and the area is $\mathcal{O}((|V| + |E|)^2 \cdot k)$. For the 3D drawing the same bounds hold. The 2D drawing has a width and height of $\mathcal{O}((|V| + |E|)^2)$ and an area of $\mathcal{O}((|V| + |E|)^4)$.*

Proof. Let $\mathcal{S} = \{S_1, \ldots, S_r\}$ be the set of all SCCs. Let B_i be the number of blocks in S_i and N_i be the number of (dummy) vertices in S_i. The width of the compacted drawing of S_i is at most B_i. The height of the drawing is the sum of the height of all blocks at most and, hence, at most N_i. Shearing this drawing with slope $-\text{wind}(S_i) \cdot k/\text{width}(S_i)$ adds at most $(N_i \cdot B_i)/k$ to the width. As $N_i \leq B_i \cdot k$, the width of the drawing of S_i is in $\mathcal{O}(B_i^2)$. The width of the drawing of G is at most the sum of the widths of the drawings of all SCCs and, thus, in $\mathcal{O}((|V| + |E|)^2)$. As the height is k, the area is in $\mathcal{O}((|V| + |E|)^2 \cdot k)$.

The height and width of the 2D drawing is twice the width of the intermediate drawing and the area is $\mathcal{O}((|V| + |E|)^4)$. □

Although, the width can be quadratic, our benchmarks in Chapter 7 show that the width to expect in drawings is much smaller. The worst case can occur, however, if, e. g., the graph $G = (V, E, \phi, <)$ consists of one SCC only and the cutting of the SCC wraps around the center several times. Then it is possible that the traversal uses $\Theta(|V^p|)$ intra block edges which results in a compacted drawing of height $\Theta(|V^p|)$. Consider Figure 5.10 for an example. Note that the drawing does not contain any crossing and each vertex has at most one incoming and one outgoing edge. Hence, no edge is removed during the block building phase. Ignoring the dashed edges for now, the graph consists of two sequences of edges starting at the vertices a and b. These sequences wrap around the center three times until they reach the vertices c and d. Note that these two sequences use twelve segments each. Together with the two dashed edges they build up one large ring: Starting from a traverse in edge direction until c is reached going one vertex towards the center each time an end vertex is reached. From c traverse to d and from there against edge direction towards b (again going inwards each time an end vertex is reached). Then, use the two dashed edges in edge direction to reach a again. As the cutting of an SCC cuts each ring exactly once, the compacted drawing has at least the height twelve as at least one of the sequences from a to c or from d to b is not cut. The width of the compacted drawing is at least ten, as all ten blocks are part of the ring. Adding two more edges with span three starting between the vertices c and d increases the width of the compacted drawing by two and the height by three. Hence, in this case, the height and the width of the compacted drawing are direct proportional to the size of the graph G. As the slope to use is inverse proportional to the width of the compacted drawing, the width of the intermediate drawing grows quadratically in the size of the graph G. Hence, in the 2D drawing, the width and height grow quadratically.

Note that the width of the drawing reduces to $\mathcal{O}(|V|+|E|)$ if there are no complex SCCs in the graph. This reduces the area of the intermediate and 3D drawings to $\mathcal{O}((|V| + |E|) \cdot k)$ and of the 2D drawing to $\mathcal{O}((|V| + |E|)^2)$. Complex SCCs can always be avoided by using our global sifting heuristic and a modified block building algorithm as we show in the next section.

Theorem 5.4. *The layout algorithm described in Algorithm 5.1 has a time complexity of $\mathcal{O}(|V^p| + |E^p|)$ for a proper ordered cyclic k-level graph $G^p = (V^p, E^p, \phi^p, <)$.*

Proof. Flipping the input graph horizontally and/or vertically can trivially be done in $\mathcal{O}(|V^p|+|E^p|)$. Building the cyclic block graph consists of two parts: Marking all type 1 conflicts has a time complexity of $\mathcal{O}(\deg^-(v))$ for one vertex and $\mathcal{O}(|V^p| + |E^p|)$ in total. Aligning each (dummy) vertex with a median neighbor is done in $\mathcal{O}(|V^p|)$. The size of the cyclic block graph is $\mathcal{O}(|V^p|)$ as it contains $\mathcal{O}(|V^p|)$ (dummy) vertices, intra block edges, and inter block edges each. Splitting long blocks results in traversing each block completely and a time complexity of $\mathcal{O}(|V^p|)$. Computing the strongly connected components of the cyclic block graph can be done in linear time $\mathcal{O}(|V^p|)$ [144]. Cutting an SCC, assigning new levels ϕ', compacting, and shearing an SCC has a time complexity linear in the size of the SCC and of $\mathcal{O}(|V^p|)$ in total

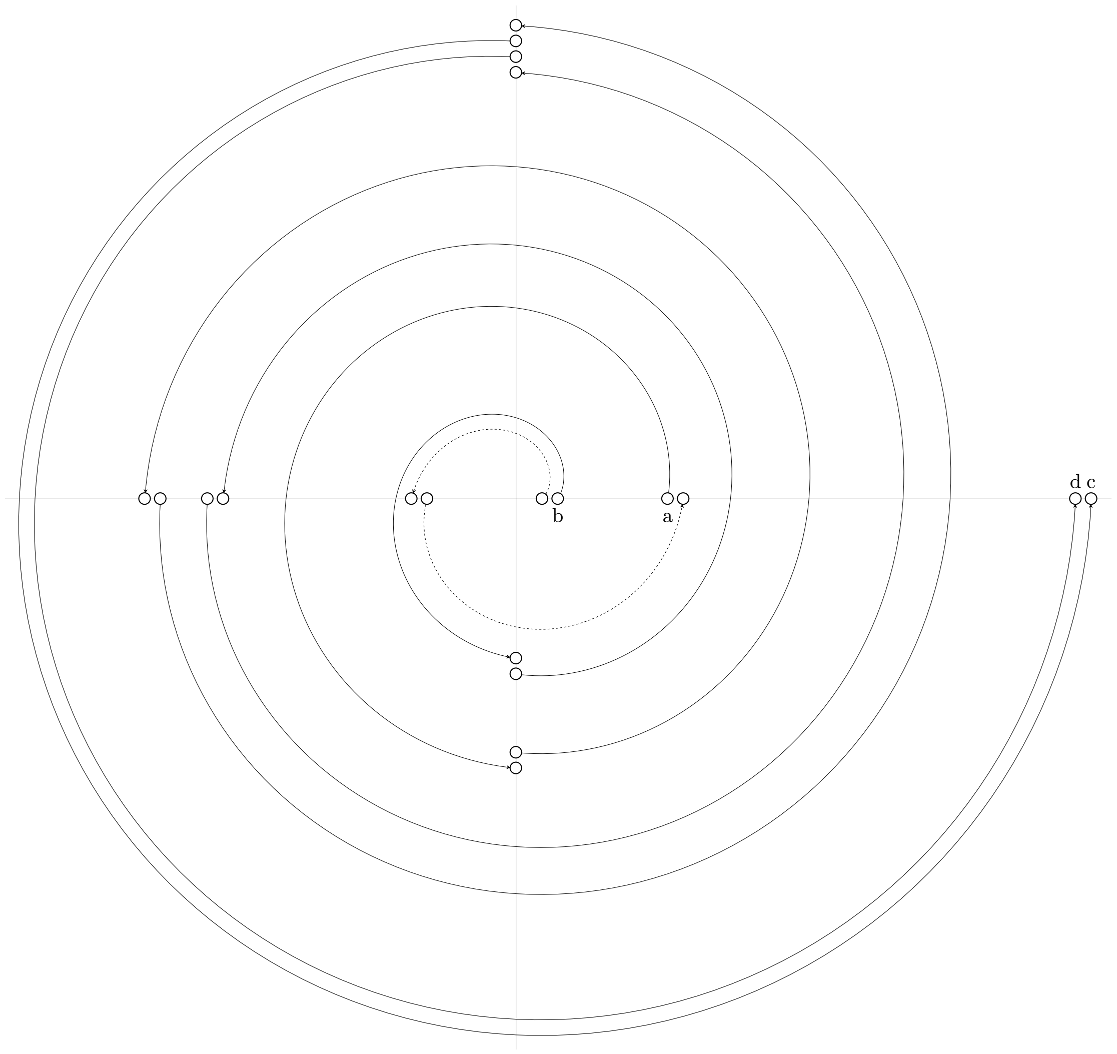


Figure 5.10. Graph using quadratic width and height in the 2D drawing

for all SCCs. Finally, compacting G^p is done in linear time as well. All these steps are carried out four times. Balancing the four results is trivially done in $\mathcal{O}(|V^p|)$. $\square$

5.5 Avoiding Cyclic Dependencies

The rings, i. e., the cyclic dependencies in the left-to-right order in cyclic level graphs result in sheared inner segments. This gives symmetric layouts at the cost of a potentially quadratic width of the drawing. Trying to avoid these dependencies in the coordinate assignment phase is too late as they are generated during the crossing reduction. Figure 5.4 shows a cyclic level graph which has cyclic dependencies using the optimal permutations on each level. Hence, an optimal cyclic crossing reduction or an cyclic level planarity testing and embedding algorithm finds these permutations and creates cyclic dependencies.

In Chapter 4, we presented our global sifting algorithm, which can be used for cyclic crossing reduction. There, each original vertex and each chain of inner segments is given a distinct x-coordinate. These subgraphs are called blocks as well. Using these x-coordinates results in a drawing with all inner segments aligned vertically. Thus, there are no cyclic dependencies between these blocks. However, our cyclic coordinate assignment phase (as well as [19]) uses larger blocks than the global sifting heuristic. The algorithm tries to align as many (dummy) vertices as possible to larger blocks. But this can result in cyclic dependencies again.

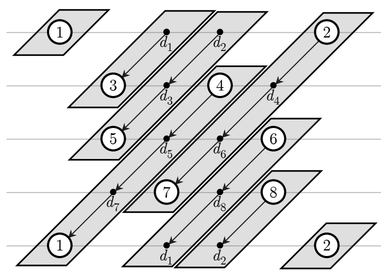
When using the global sifting heuristic as a crossing reduction, there are two options which blocks to use in our coordinate assignment algorithm. Using the *minimal blocks* of the crossing reduction results in no cyclic dependencies, vertical inner segments, linear width of the drawing but completely unbalanced drawings as only dummy vertices are aligned. Using the *maximal blocks* described in this chapter places vertices balanced with respect to their neighbors. However, there may be cyclic dependencies which result in sheared inner segments and potentially quadratic width.

Examine the intermediate drawings of a cyclic 4-level graph in Figure 5.11 for an example. It consists of four edges spanning three levels each. Using the maximal blocks as described in this chapter, the graph has cyclic dependencies, e. g., $x(1) < x(d_1) = x(3) < x(d_3) = x(5) < x(d_5) = x(d_7) = x(1)$, see Figure 5.11(a). The vertices are aligned, but the edges have to be sheared. However, when using the minimal blocks of our global sifting algorithm, no such rings exist. Figure 5.11(b) shows such a drawing with all inner segments vertical, but less aligned vertices.

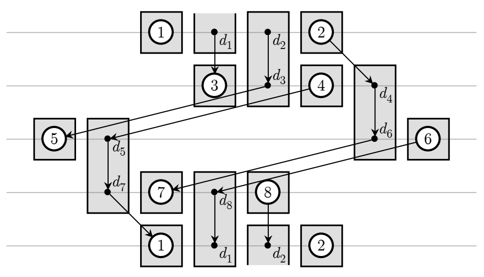
Obviously, using smaller blocks, if larger ones would not cause a cyclic dependency, only corrupts the balance without any benefit. Hence, starting with the maximal blocks and splitting blocks into smaller ones only if they are part of a cyclic dependency seems sensible. Consider Figure 5.11(a) again. When using maximal blocks, all eight outer segments are used for aligning. However, removing four of them destroys all cyclic dependencies already. One possible result is shown in Figure 5.11(c): The four outer segments in blocks result in the alignment of vertices 1 and 2 in contrast to Figure 5.11(b). The vertices 3 and 8 were already aligned in Figure 5.11(b) by coincidence.

The exact problem is to find the minimal set of intra block edges of outer segments whose removal results in an acyclic block graph. Finding such a minimum set in an arbitrary graph corresponds to the $\mathcal{NP}$-hard feedback arc set problem [69, 92] and has been discussed in Section 2.1 for the decycling phase of the hierarchical Sugiyama framework. But a cyclic block graph has a very regular structure: Each vertex has at most two incoming and two outgoing edges forming a grid with edges pointing to the right and downwards. Furthermore it is planar and all rings wrap around the center in the same direction. The feedback arc set problem in planar graphs can be solved in polynomial time [102]. For the special case of removing all clockwise or counter-clockwise cycles in a fixed embedding, an $\mathcal{O}(|V| \log |V|)$ time algorithm has been found [136].

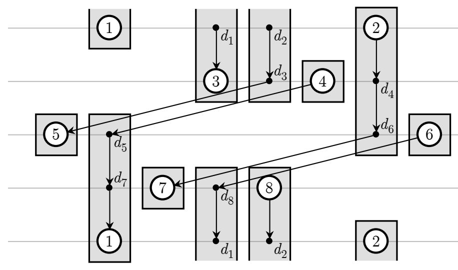
However, we only want to remove intra block edges of outer segments to destroy



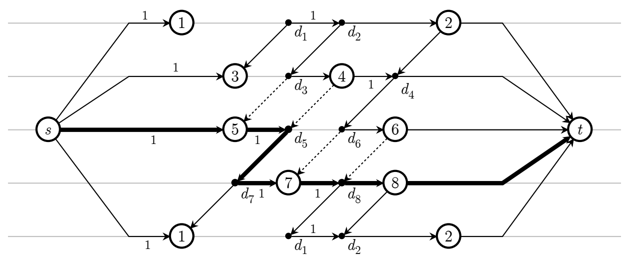
(a) Drawing with maximal blocks



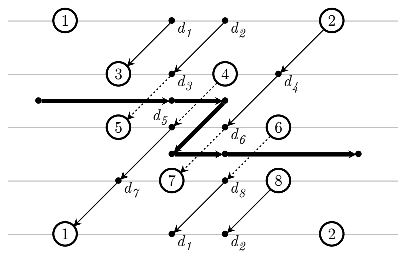
(b) Drawing with minimal blocks



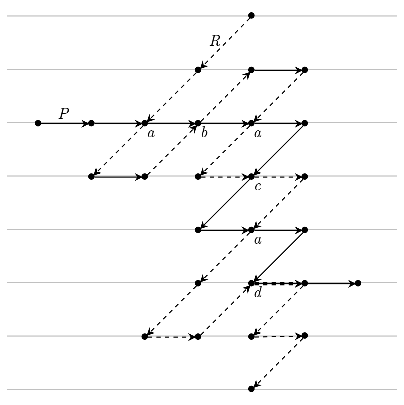
(c) Drawing with largest blocks without cyclic dependencies



(d) Outer segments graph



(e) Interpretation as dual graph



(f) Crossings of paths and rings

Figure 5.11. Finding the minimal set of intra block edges to delete

all rings. Furthermore, rings can traverse intra block edges in both directions. In this special case, we can find the minimal number of those edges in linear time. The idea is similar to the cutting of an SCC. However, this time we cut intra block edges instead of inter block edges and we want to cut as few as possible. We deal with each SCC (using maximal blocks) separately.

Definition 5.4. *Let S be an SCC with* wind$(S) = 1$. *The* outer segments graph S' *of S consists of all vertices of S and two new vertices s and t. Each rightmost vertex of S is connected to t with a directed edge of length 0. S' contains all intra block edges of S. These edges have length 0. For a vertex v of S let e be its incoming inter block edge if it exists and (s, v) otherwise. If v has an incoming outer segment, S' contains e with length 1. If v has an incoming inner segment, S' does not contain e. Finally, if v has no incoming segment, i. e., no incoming intra block edge, S' contains e with length 0.*

We now prove that the length of the shortest directed path from s to t in the outer segments graph gives the number of intra block edges to delete. For each used edge (u, v) with length 1, we remove the incoming intra block edge of v. Figure 5.11(d) shows the resulting outer segments graph for the graph in Figure 5.11(a) where edge lengths 0 are not shown. Each vertex with an incoming outer segment has an incoming edge from the left with length 1. The shortest path from s to t is drawn in bold. The vertices $5, d_5, 7$ and d_8 are entered from the left and have an incoming outer segment. These four outer segments are dashed and are removed in Figure 5.11(c). Note that drawing of the outer segments graph in Figure 5.11(d) contains duplicates of horizontal edges on the first and last level as the corresponding vertices are drawn two times. The graph itself has the same cyclic structure as the cyclic block graph, however.

We want to give some intuition regarding the outer segments graph here. A outer segments graph is (apart from s and t) a subgraph of the cyclic block graph. Both have a regular grid structure: Each vertex has at most one incoming edge from above and one from the left, one outgoing edge downwards, and one outgoing edge to the right. A path from s to t in the outer segments graph is allowed to traverse in edge direction only, i. e., downwards and to the right. A ring in the cyclic block graph can use intra block edges in both directions and inter block edges in their direction only. Hence, a ring can traverse upwards, downwards and to the right in the cyclic block graph.

Intuitively, we remove the incoming intra block edge each time the path from s to t in the outer segments graph enters a vertex from the left. We can only do so if the edge from the left exists, which is the case only if the intra block edge from above represents an outer segment. As these edges are the only ones with length 1, they are minimized in a shortest path from s to t.

This method has some similarities to [136] which uses the dual graph of the input graph to solve the feedback arc set problem in planar graphs. Due to the regular grid structure of the block graph, the outer segments graph is very similar to the dual graph of the block graph. Each vertex v of the outer segments graph can be interpreted as the vertex of the dual graph belonging to the face of the grid above the right outgoing edge of v. In Figure 5.11(e) the path of Figure 5.11(d) is shifted to represent a path in the dual graph. Now, the path crosses exactly the outer segments we remove. We use the dual graph as an intuition only as the proofs benefit from having the same vertex set for the block graph and the outer segments graph.

Lemma 5.7. *Let S be a complex SCC of a cyclic block graph using maximal blocks such that there is no ring when using minimal blocks. Let $\operatorname{wind}(S) = 1$. Let P be a path from s to t in the outer segments graph. Let Q be the set of vertices on P, which are entered from the left by P. Removing the incoming intra block edge of each vertex $q \in Q$ from S results in an acyclic block graph.*

Proof. Let R be a simple ring in S. As $\operatorname{wind}(S) = 1$, $\operatorname{wind}(R) = 1$ holds. As R is a circle in the cyclic block graph it splits the block graph in a left and a right part. As P is a path from a leftmost to a rightmost vertex, R and P have at least one common vertex. Let v be the first common vertex on P. Hence, the predecessor of v in P lies in the left part of the block graph. Note that P can traverse to the right and downwards only and R can traverse to the right, upwards, and downwards only. P and R cannot enter v using the same edge as then v is not the first common vertex. Figure 5.11(f) contains the remaining four cases. If P enters v from above and R from the left (case c) or from below (case d) or if P enters from the left and R from below (case b) then the path P has already been in the right part of the block graph. Hence, this cannot be the first crossing. The remaining case (case a) is that P enters from the left and R from above. Then the incoming intra block edge of v is one of the removed edges and R does not exist in the new block graph. ☐

Lemma 5.8. *Let S be a complex SCC of a cyclic block graph using maximal blocks such that there is no ring when using minimal blocks. Let $\operatorname{wind}(S) = 1$. Let M be a minimal set in terms of inclusion of intra block edges, whose removal results in an acyclic block graph. Then, there exists a path in the outer segments graph from s to t using $|M|$ edges of length 1.*

Proof. Let all segments of M be removed from S. S is acyclic then. Let $L = v_1, \ldots, v_{|M|}$ be the set of lower end vertices of the outer segments in M sorted in topological order of their minimal blocks. We search for a path from s to t by traversing all vertices of L in this order and using edges of length 1 only to enter a vertex v_i ($1 \leq i \leq |M|$). We construct that path piecewise from each v_i backwards.

The vertex v_1 has an incoming edge from the left. We traverse that edge against its direction to enter a new block B. We traverse the intra block edges of B against their direction to the topmost vertex and repeat this procedure. We cannot enter v again as we would have traversed a ring backwards then. Hence, we have to reach s eventually. Reversing this path gives a path from s to v_1. Starting at a vertex v_i ($i > 1$) gives a similar result. We have to reach v_{i-1} eventually. If we reach v_i first, we have found a ring. If we reach v_j ($j < i - 1$), deleting the incoming outer segments of v_{j+1} to v_{i-1} would have been unnecessary. This is a contradiction to the minimality of M. We cannot reach a vertex v_j $(j > i)$ either as we traverse the outer segments graph against its topological order. A similar argument shows that a path against the edge directions from t to $v_{|M|}$ exists. Reversing and concatenating all these paths gives a path $s, \dots, v_1, \dots, v_2, \dots, \dots, \dots, v_{|M|}, \dots, t$ using $|M|$ edges of length 1. □

Theorem 5.5. *Let S be a complex SCC of a cyclic block graph using maximal blocks such that there is no ring when using minimal blocks. Let P be a shortest path from s to t in the outer segments graph. Let Q be the set of vertices on P, which are entered from the left by P. Removing the incoming intra block edge of each vertex $q \in Q$ from S is a minimal set of intra block edges to remove from a cyclic block graph to destroy all rings.*

Proof. Lemma 5.7 proves that removing the incoming intra block edges for each vertex $q \in Q$ results in a acyclic block graph. Lemma 5.8 shows that for each minimal set of such edges a corresponding path exists. Hence, the shortest path gives the minimal set of edges to delete. □

Finding the shortest path from s to t with edge lengths 0 and 1 only is obviously possible in linear time with a slightly modified breadth first search.

5.6 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 hierarchical 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. Drawing all inner segments vertically is not always possible in the cyclic case as there can be cyclic dependencies in the left-to-right order of the vertices. We solve this problem by shearing the drawing. Hence, the symmetry of a graph can be represented in the drawing. However, the width of the drawing may be quadratic then. Combining our global sifting crossing reduction and our cyclic coordinate assignment phase avoids these cyclic dependencies at the cost of slightly less balanced drawings.

5.7 Open Problems

Although the algorithm by Brandes and Köpf is the standard algorithm for the coordinate assignment phase in the hierarchical Sugiyama framework, several other heuristics exist. Further research is necessary to determine which of them can be transferred to the cyclic case. Methods using a minimization problem, e. g., like Gansner et al. [66], should be adaptable in a straight forward way.

When using one run only, less different slopes are used, but each vertex is aligned with its outgoing median only. On the other hand, shearing the drawing can be avoided completely as described in Section 5.5. A more detailed analysis of which combination of the number of runs and the avoidance of cyclic dependencies gives the best result is needed.

In our version using only one run, we compact always to the right. This less-than-ideal solution corrupts the symmetry of the drawing. Here, a symmetric compaction is preferable.

The idea of drawing all edges in a strongly connected component of the block graph with the same slope can lead to quadratic width in the size of the graph including dummy vertices. Therefore, a trade-off between the number of used slopes and the width of the resulting drawing could lead to nicer drawings.

5.8 Examples of Drawings

In this section, we give example outputs of our cyclic coordinate assignment algorithm. The input graph G_{124} has 124 original vertices and 189 edges. As the leveling on twelve levels introduced 61 dummy vertices, the graph has 250 segments. Figure 5.12 presents the final cyclic drawing of G_{124} where dummy vertices were already replaced by bends. Figure 5.13 shows the intermediate drawing of G_{124} with dummy vertices. This is also the surface of the cylinder of the 3D drawing of G_{124}. Figure 5.14 and Figure 5.15 contain the same graph drawn with only one balanced run as suggested in Chapter 5.3.3. It is clearly visible in the intermediate drawing that this produces less different slopes. In the cyclic drawing, however, this difference is much more difficult to recognize.

Finally, Figure 5.16 and Figure 5.17 illustrate the cyclic drawing of a smaller graph G_{24}, which consists of 24 vertices and 44 edges. Due to the 76 dummy vertices, G_{24} has 120 segments. The drawing is again produced by only one balanced run. In Figure 5.17 it is clearly visible that at least two complex SCCs in the cyclic block graph exist. The edges on the left hand side are sheared with a different slope than the edges in the middle of the drawing.

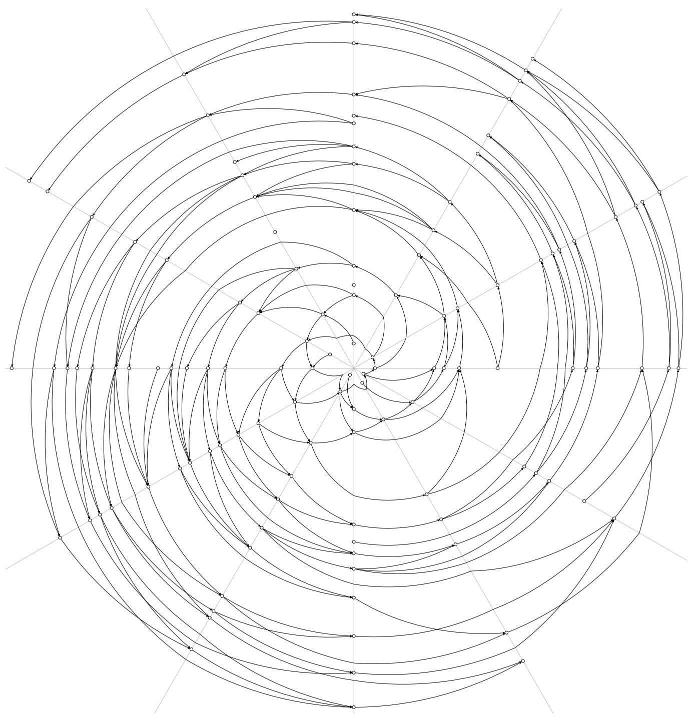


Figure 5.12. 2D drawing of G_{124} with 124 vertices, 189 edges and 12 levels

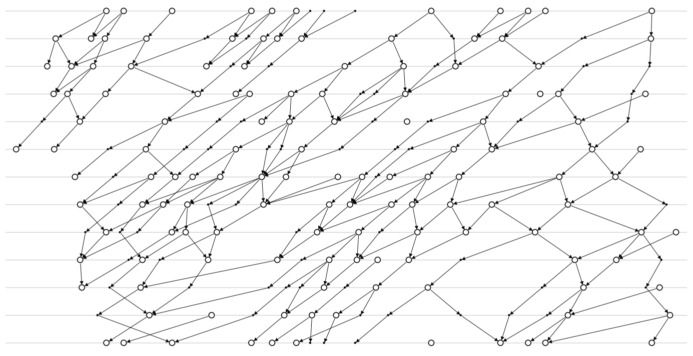


Figure 5.13. Intermediate drawing and surface of the 3D drawing of G_{124}

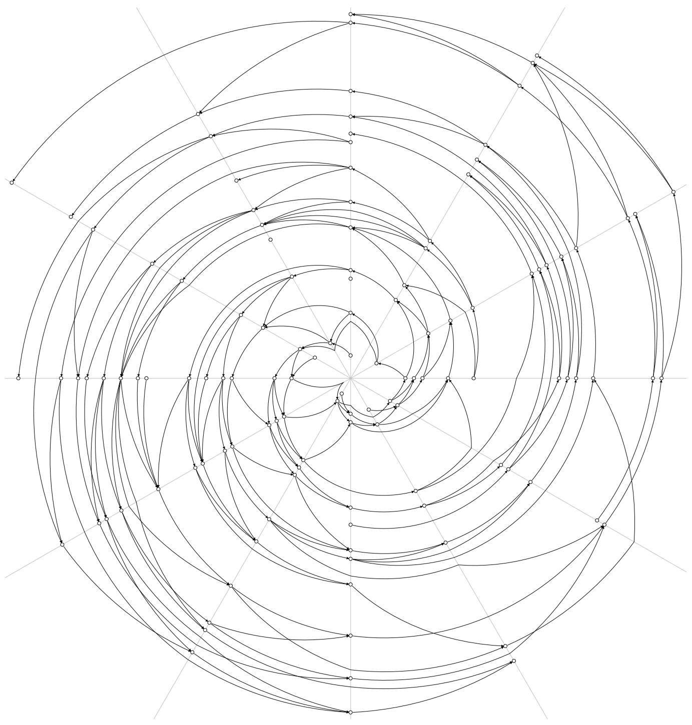


Figure 5.14. 2D drawing of G_{124} produced by one balanced run

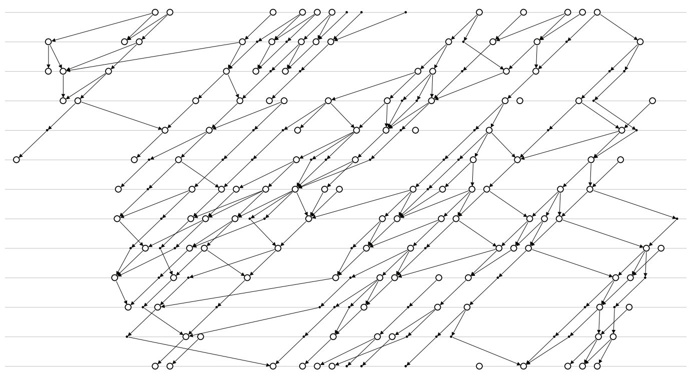


Figure 5.15. Intermediate drawing, surface of the 3D drawing of G_{124} (one run)

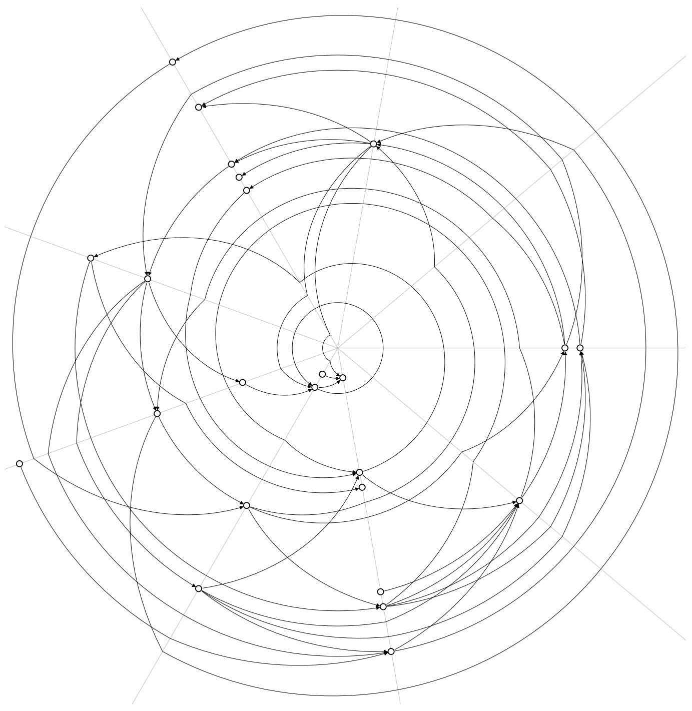


Figure 5.16. 2D drawing of G_{24} produced by one balanced run

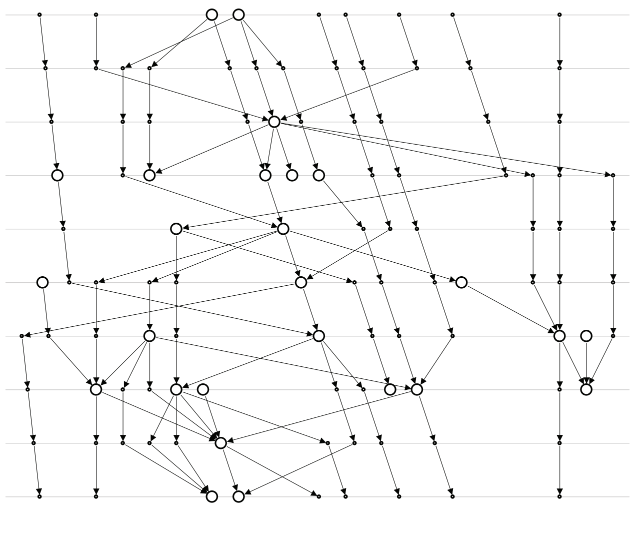


Figure 5.17. Intermediate drawing and surface of the 3D drawing of G_{24} (one run)

6 Cyclic Level Planarity

In this chapter, we treat cyclic level planarity. A (cyclic) level planarity testing and embedding algorithm can be integrated into the Sugiyama framework as a special crossing reduction. The input is a (cyclic) level graph and the output is an ordering of the vertices on each level which does not cause any crossings if such an ordering exists. Note that minimizing the number of crossings in a (cyclic) level graph is $\mathcal{NP}$-hard (see [70] and Chapter 4.2). However, testing whether a level graph can be drawn without any crossings at all is solvable in linear time. In this chapter we show that cyclic level planarity can be tested in linear time as well for the special case of strongly connected graphs. We first present results regarding planarity in Chapter 6.1 and level planarity in Chapter 6.2. After basic definitions in Chapter 6.3, we present cyclic level non-planarity pattern and a cyclic level planarity testing and embedding algorithm in Chapter 6.4. Finally, we give a summary in Chapter 6.5 and open problems in Chapter 6.6.

6.1 Planarity

Regarding planarity without levels, several very old results exist. Euler showed that for any planar graph $G = (V, E)$ with at least 3 vertices $|E| \leq 3|V| - 6$ holds. Kuratowski proved in 1930 [94] that a graph is planar if and only if its underlying undirected graph does not contain a subgraph homeomorphic to the K_5 or $K_{3,3}$. The K_5 is the undirected complete graph with five vertices and the $K_{3,3}$ is the undirected complete bipartite graph with three vertices in each set. These graphs are called *non-planarity patterns* or *Kuratowski subgraphs*. Fáry showed in 1948 [55] that each planar graph can be drawn with edges as straight lines without crossings.

Nowadays, several algorithms to test the planarity of a graph in linear time exist,

e. g., the path addition method by Hopcroft and Tarjan [82] and methods by Shih and Hsu [135] as well as Boyer and Myrvold [18]. The vertex addition method of Lempel, Even, and Cederbaum [99] is another planarity testing method. One important step to improve their method to linear time was the inclusion of the PQ-Tree data structure by Booth and Lueker [17]. All these algorithms can be used to construct a planar embedding if the graph is planar and to find a Kuratowski subgraph if the graph is non-planar. There are several algorithms to produce straight-line drawings of planar graphs on quadratic area, e. g., [31, 91, 131].

6.2 Level Planarity

Deciding whether a directed planar graph is drawable such that all edges point upwards and have no crossings is an $\mathcal{NP}$-hard problem [71]. However, fixing the y-coordinates of each vertex yields a polynomial problem called *level planarity*.

Di Battista and Nardelli gave a linear time level planarity testing and embedding algorithm for proper hierarchies [33]. In a hierarchy, all vertices without incoming edges are placed on the first level. Heath and Pemmaraju [79] proposed a linear time level planarity testing algorithm for arbitrary proper level graphs, which is not correct, however [87]. Jünger et al. [88] gave the first linear time level planarity testing algorithm for arbitrary non-proper level graphs. They improved it to compute a level planar embedding later [86]. All these algorithms use the PQ-tree data structure [17] to store the set of possible permutations on the current level.

Recently, much simpler algorithms to test level planarity of arbitrary proper level graphs in quadratic time have been proposed. The main idea of these algorithms is that for two edges (u, v) and (w, x) $(u \neq w, v \neq x)$, spanning the same levels, u is left of w in a level planar drawing if and only if v is left of x. Randerath et al. [124] use a logical term in 2CNF formulation to model this problem where each variable determines the ordering of two vertices on the same level. Healy et al. [76, 77] construct a new graph called vertex exchange graph. Here each vertex represents a pair of vertices of the input graph on the same level. Two vertices of the vertex exchange graph are connected if their original vertices are connected by two edges. If these edges cross in an initial drawing the edge is marked with "-", otherwise with "+". The input graph is level planar if and only if the vertex exchange graph does not contain a circle with an odd number of edges marked with "-".

Concerning level non-planarity patterns, Di Battista and Nardelli [33] presented three patterns for hierarchies, i. e., *hierarchical level non-planarity patterns* (HLNP). However, these pattern are not disjoint and removing one edge of them does not yield a level planar graph in each case, i. e., they are not minimal. Healy et al. [78] gave seven *minimal level non-planarity patterns for hierarchies* (MHLNP). This complete set consists of two tree patterns T1 (Figure 6.1(a)) and T2 (Figure 6.1(b)), a level non-planar circle C0 (Figure 6.1(c)), and four level planar circles with one (C1, Figure 6.1(d)) to four (C4, Figure 6.1(h)) paths starting from the circle (C2 having two subcases). For the formal definition of these patterns see [78].

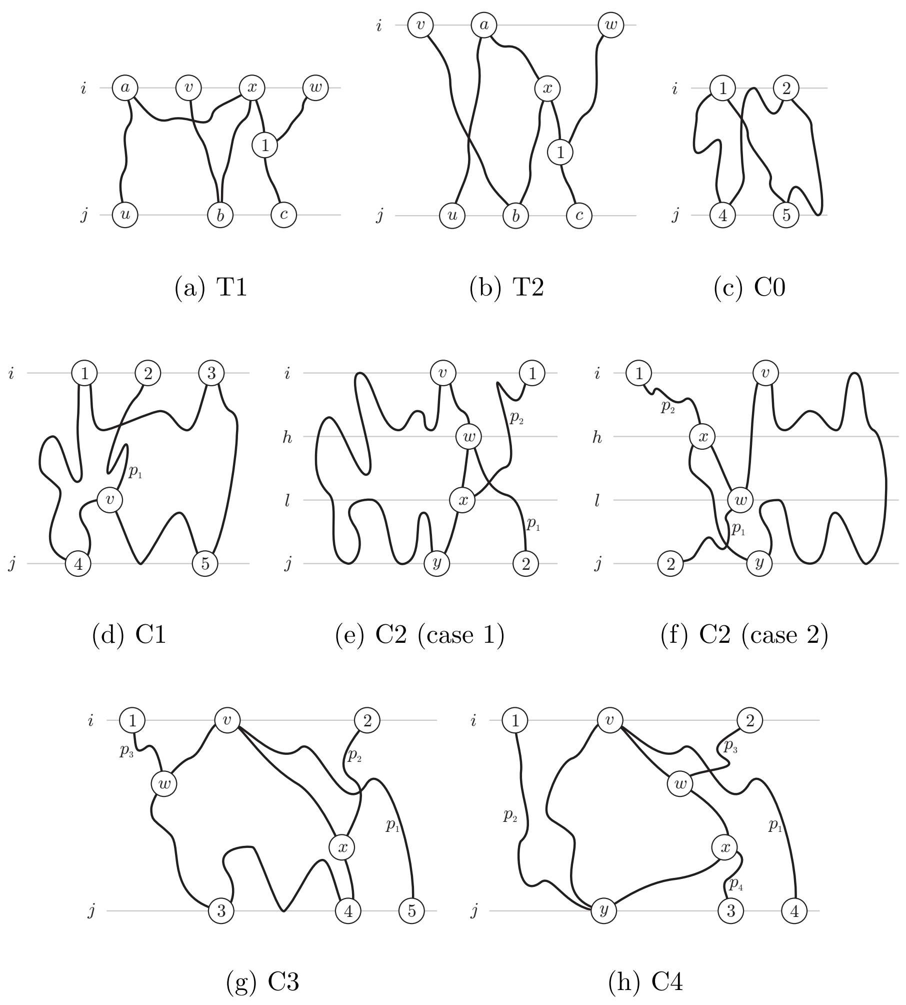


Figure 6.1. The minimal level non-planarity patterns for hierarchies (MHLNP)

Recently, there have been some results concerning the completeness of these patterns in the general level graph case. Fowler and Kobourov [61] showed that the MHLNP set is not complete for general level graphs and added two more tree patterns. Later they showed that there are infinitely many patterns in the general case [53]. The original MHLNP patterns are still complete for hierarchies, however.

Concerning the area of a drawing of a level graph, the height is determined by the number of levels k which is at most $|V|$. When allowing at most two bends per edge, e. g., by using the algorithm by Brandes and Köpf [19], the width of the drawing is at most $\mathcal{O}(|V|)$. Eades et al. proved that a straight-line drawing of a level planar graph is always possible [42]. However, this can lead up to exponential width [35].

We extend level planarity to cyclic level planarity in this chapter. The extension to radial level planarity has already been studied [6]. In a radial level drawing, levels form concentric circles and the edges are drawn outwards.

6.3 Preliminaries

A *(cyclic) level planar embedding* $\mathcal{G}$ of a (cyclic) level graph $G = (V, E, \phi)$ consists of two adjacency lists $N^-(v)$ and $N^+(v)$ for each vertex $v \in V$ which contain the end vertices of incoming and outgoing edges, respectively. Both lists are ordered from left to right. A *hierarchy* is a level graph G s. t. each vertex v with $\phi(v) \neq 1$ has an incoming edge.

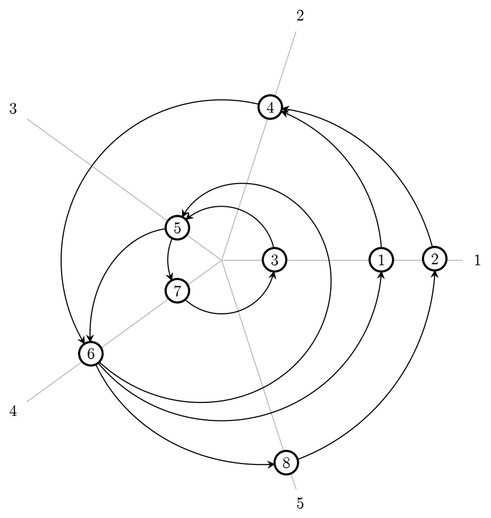


Figure 6.2. Drawing of a cyclic level graph G

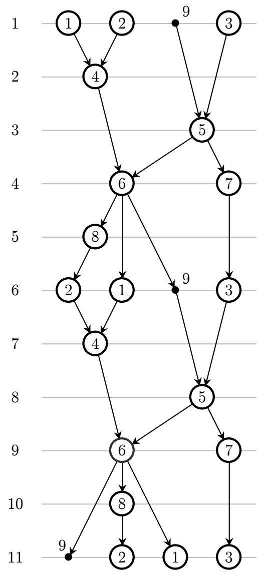
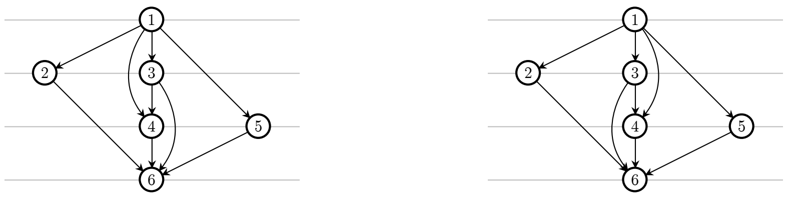


Figure 6.3. $(2, 1)$-hierarchy of G

Note that in this chapter we do not assume the graph to be proper. As there are no dummy vertices on long edges, the permutation of each level does not suffice to determine an embedding. See Figure 6.4 for two embeddings with different faces of the same graph with the same order of vertices on each level. Hence, we use sorted adjacency lists of incoming and outgoing edges.



(a) Drawing 1 (b) Drawing 2

Figure 6.4. Different faces with the same order of vertices on each level

On the other hand, using sorted adjacency lists is only sufficient in the context of planarity. Otherwise, the routing of a long edge would not be specified. Hence, it is not possible to use sorted adjacency lists in the previous chapters.

6.4 Cyclic Level Planarity

As a first approach towards cyclic level planarity, we proposed a $\mathcal{O}(|V| \log |V|)$ cyclic level planarity testing and embedding algorithm for strongly connected graphs [9]. This approach has similarities to the planarity testing algorithms by Hopcroft and Tarjan [82] and Boyer and Myrvold [18]. The idea is to remove an arbitrary simple cycle from the input graph. Each resulting connected component then placed either on the left hand side or the right hand side of the cycle. To do so already placed components may be flipped to the other side. The algorithm presented in this chapter is easier to implement and has a linear time complexity, however. Both can use any existing level planarity testing and embedding algorithm for hierarchies as a black box. We first consider cyclic level non-planarity patterns and derive the algorithm from them thereafter.

6.4.1 Cyclic Level Non-planarity Patterns

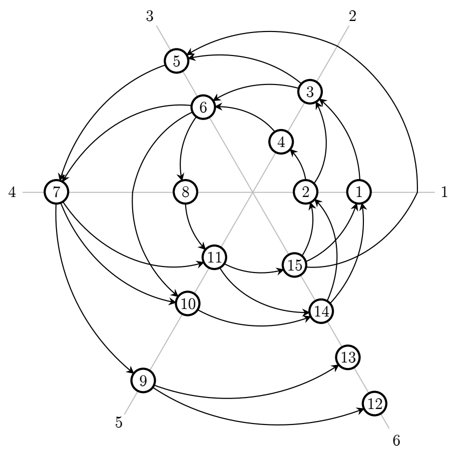
In this section, we give a characterization of cyclic level non-planarity patterns in strongly connected graphs (SCLNP).

Definition 6.1. *A* (cyclic) level non-planarity pattern *P is a set of (cyclic) level non-planar graphs with structural similarities. We call a pattern* minimal *if for each element of P the removal of one edge makes the graph (cyclic) level planar. A (cyclic) level graph G* matches *a pattern P if there exists a graph $p \in P$ s. t. p is a subgraph of G. A set of patterns S is minimal if each pattern in S is minimal. S is* complete *if each (cyclic) level non-planar graph matches a pattern in S.*

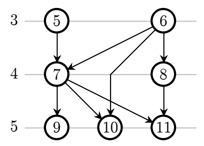
The term “structural similarities” seems to be rather unfitting for a formal definition. However, the situation regarding level non-planarity patterns is much more complicated than the non-planarity patterns of Kuratowski [94]. The (infinite) set of all minimal non-planar graphs can be naturally divided into two sets: One containing all such graphs homeomorphic to the K_5 and one set with all graphs homeomorphic to the $K_{3,3}$.

However, in the field of level planarity things are different. The structure of the graph itself and its leveling determine whether it is planar or not. The (infinite) set of all minimal level non-planar graphs is well-defined. But the problem arises how to divide this set into patterns. Patterns cannot be described by homeomorphic subgraphs anymore. In level graphs, patterns are described by paths between certain levels. Di Battista and Nardelli [33] presented three patterns for hierarchies (HLNP). This set is complete but not minimal. Healy et al. [78] gave a complete set of seven minimal level non-planarity patterns for hierarchies (MHLNP). This alone shows that there are different approaches to combine level non-planar graphs to patterns. Regarding the patterns of Healy et al., the question arises whether $T1$ and $T2$ should be merged to one pattern or if the two cases of $C2$ justify to split the pattern into two separate ones. But in the end, only the set of all minimal (cyclic) level non-planar graphs is important. How we split this set into patterns and how many patterns arise from that process is secondary. Hence, splitting the subgraphs into patterns is more or less a matter of personal taste and explains the term "structural similarities" in the definition. This discussion shows that level non-planarity patterns are still much less understood than non-planarity patterns.

Level non-planarity patterns and cyclic level non-planarity patterns are very similar. Examine Figure 6.5 for an illustration. Figure 6.5(a) shows the example graph G^* which is cyclic level non-planar. To see that consider Figure 6.5(b) which shows a subgraph H of G^* consisting of all vertices on the levels 3 to 5 and all edges between them. The vertices on each level of H cannot be permuted to avoid all crossings of H. Hence, H is level non-planar. But then there is no possibility to permute the levels 3 to 5 in G^* to get rid of the crossings between the levels 3 and 5 in G^* as well. G^* has at least these crossings and is cyclic level non-planar then. Consequently, searching for level non-planarity patterns in acyclic subgraphs of cyclic level graphs is a reasonable way to prove cyclic level non-planarity. To formalize this idea, we use the following definition.



(a) Cyclic drawing of G^*



(b) Acyclic Subgraph H of G^*

Figure 6.5. Level non-planar subgraph of a cyclic level graph

Definition 6.2. *Let $G = (V, E, \phi)$ be a cyclic k-level graph. Let $c \in \mathbb{N}$ $(c > 0)$ and $l \in \{1, \ldots, k\}$. Suppose that no edge crosses level l (if such edges exist, add a vertex d for each such edge $e = (u, v)$ to level l, remove the edge e and add the edges (u, d) and (d, v) to the graph). The* (c, l)-hierarchy *H of G is a $(ck + 1)$-level hierarchy.*

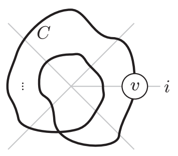


Figure 6.6. The additional cyclic level non-planarity pattern (CC)

For each vertex v on level l, H has $c+1$ duplicates $v^{(1)}, v^{(k+1)}, \dots, v^{(ck+1)}$ with $v^{(i)}$ on level i. Let $w \in V$ be a vertex with $\phi(w) \neq l$ and let $l' = \phi(w) - l + 1$ if $\phi(w) > l$ and $l' = \phi(w) - l + k + 1$ otherwise. For each such vertex w, H has c duplicated vertices $w^{(l')}, w^{(k+l')}, \dots, w^{((c-1)k+l')}$ with $w^{(i)}$ on level i. For each edge $e = (u, v) \in E$ and for each duplicate $u^{(i)}$ in H with $i < ck + 1$, H contains the edge $(u^{(i)}, v^{(i+\text{span}(e))})$.

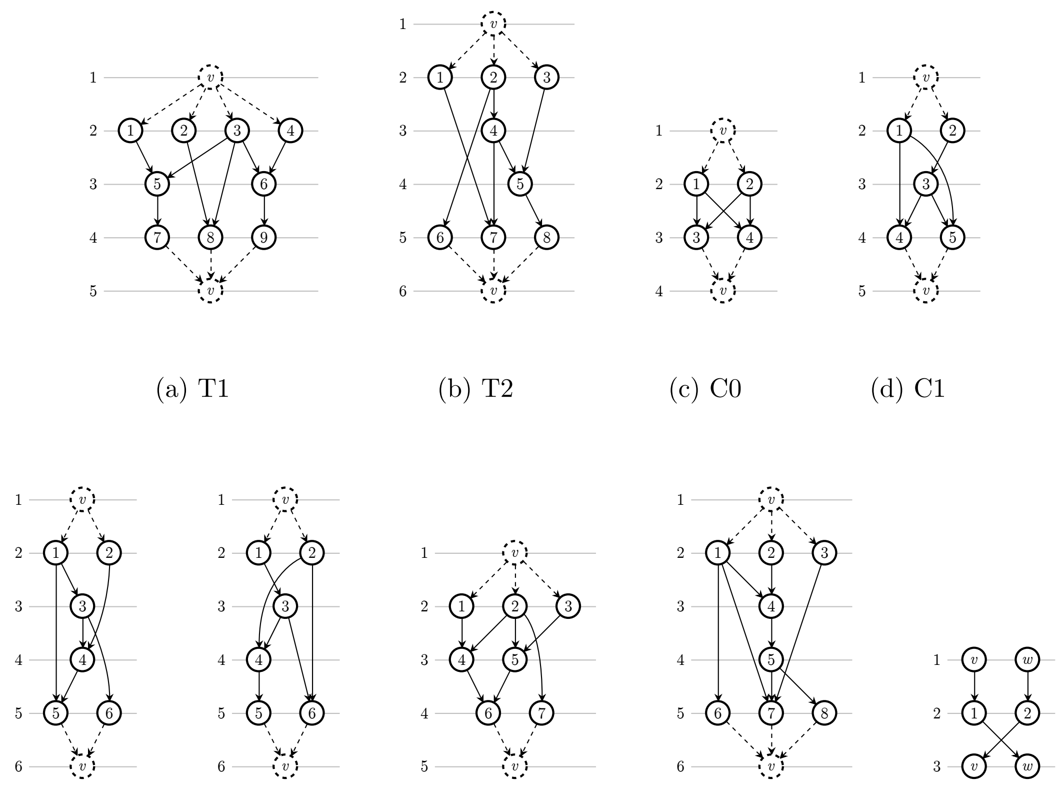
Informally speaking, the (c, l)-hierarchy H of G is obtained by splitting G at level l (thus creating a level graph) and duplicating the graph c times one below the other. In the context of a cyclic level graph $G = (V, E)$ and its (c, l)-hierarchy H, we denote an original vertex v of G without index and its duplicate on level i in H as $v^{(i)}$ in general. Figure 6.2 shows a cyclic 5-level graph and Figure 6.3 a level plane drawing of the corresponding $(2, 1)$-hierarchy.

A $(1, 1)$-hierarchy of a cyclic k-level graph G and the intermediate drawing of G can look identical. Both contain $k + 1$ levels with all vertices on the first level duplicated on the last one. However, the intermediate drawing is just a drawing of G which looks like a (non-cyclic) $k + 1$-level graph whereas the $(1, 1)$-hierarchy is a (non-cyclic) $k + 1$-level graph. Hence, in a drawing of a $(1, 1)$-hierarchy the vertices on the first and last level may have different permutations.

Lemma 6.1. *Let G be a strongly connected cyclic k-level graph. Let $0 < c \in \mathbb{N}$ and $l \in \{1, \dots, k\}$. If the (c, l)-hierarchy H of G is level non-planar, then G is cyclic level non-planar.*

Proof. We show the contrapositive. Let G be cyclic level planar. Then, using the ordering of $N^-(v)$ and $N^+(v)$ of each vertex v in G for each copy of v in H results in a level planar embedding of H. □

Hence, each level non-planarity pattern can be used to describe cyclic level non-planarity as well. However, in a hierarchy, each vertex v with $\phi(v) > 1$ has an incoming edge, whereas in strongly connected cyclic level graphs each vertex has an incoming and outgoing edge. Therefore, it is not obvious that each of the seven MHLNP patterns can occur in the cyclic case. But for each of the patterns a strongly connected cyclic level graph can be constructed. Figures 6.7(a) - (h) show intermediate drawings of strongly connected cyclic k-level graphs which match exactly one of the MHLNP patterns, where the vertices and edges not needed for the patterns are drawn dashed. We present the graphs as intermediate drawings to make it easier to compare them to the MHLNP patterns in Figure 6.1. We define a cyclic version of each of the seven MHLNP patterns:


(a) T1 (b) T2 (c) C0 (d) C1

(e) C2 (1) (f) C2 (2) (g) C3 (h) C4 (i) CC

Figure 6.7. Intermediate drawings of cyclic level graphs matching exactly one of the SCLNP patterns

Definition 6.3. *Let G be a cyclic k-level graph. We say that G matches the pattern CT1 if there exists $c \in \mathbb{N}$ such that the $(c, 1)$-hierarchy of G matches T1. We define the remaining six patterns in analogously and define the set of all these patterns SCLNP' $= \{$CT1, CT2, CC0, CC1, CC2, CC3, CC4$\}$.*

The obvious question that arises now is whether these are all patterns needed in the cyclic case as well. At least one more pattern is needed: Consider a simple cycle wrapping around the center more than once.

Definition 6.4. *Let G be a cyclic k-level graph and $c \in \mathbb{N}$ $(c > 0)$. A* c-cycle *is a simple cycle with span $c \cdot k$.*

Figure 6.7(i) shows the intermediate drawing of a strongly connected cyclic 2-level graph which is a 2-cycle. Note that this graph is cyclic level non-planar, but does not match any pattern of SCLNP'. On the other hand, none of the graphs in Figure 6.7(a) - (h) contains a 2-cycle. Therefore, an eighth cyclic level non-planarity pattern is needed.

Definition 6.5. *We define* CC *as the set of all c-cycles in cyclic k-level graphs with $c, k \in \mathbb{N}$ $(c > 1)$ (Figure 6.6). We set SCLNP = SCLNP' $\cup\{CC\}$.*

Proposition 6.1. *Let G be a cyclic k-level graph. If G matches CC, then G is cyclic level non-planar.*

Note that CC is a minimal pattern, as removing one edge from the cycle leads to a cyclic level planar graph.

The main result of this chapter is that these eight patterns are sufficient for strongly connected cyclic level graphs. To show that, we need the following definitions.

Definition 6.6. *We call a (c, l)-hierarchy H of a cyclic level graph G* strongly level planar *if it is level planar and has a level planar embedding s. t. the first and last level have the same permutation. We call such an embedding a* strongly level planar embedding.

Proposition 6.2. *Let G be a strongly connected cyclic k-level graph and let $l \in \{1, \dots, k\}$. G is cyclic level planar if and only if the $(1, l)$-hierarchy of G is strongly level planar. Let $c \in \mathbb{N}$. If the (c, l)-hierarchy of G is (strongly) level non-planar, then G is cyclic level non-planar.*

Definition 6.7. *Let A, B, C be three permutations of the same vertex set. We define the* lexicographical ordering with respect to A *on the set of permutations in the following way: If B and C are the same permutations, then they are equal in the ordering. Otherwise there is a leftmost position on which B and C have different vertices. Let b and c be the vertices on this position in B and C, respectively. We define $B < C$ if $b < c$ in A and $B > C$ if $b > c$ in A.*

Another way to interpret this ordering is as follows: The permutation A defines an ordering on an alphabet. A permutation B is then smaller than C if the word it defines is smaller than the word of C in the lexicographical ordering.

Lemma 6.2. *Let $G = (V, E)$ be a strongly connected cyclic k-level graph such that G does not match CC. Let H be the $(2, 1)$-hierarchy of G. Let H be level planar with a fixed level planar embedding $\mathcal{H}$. Then G is cyclic level planar and a cyclic level planar embedding $\mathcal{G}$ of G exists such that the permutation of level 1 in $\mathcal{G}$ is the same as the permutation of level $k + 1$ in $\mathcal{H}$.*

Proof. Assume for contradiction that such an embedding $\mathcal{G}$ of G does not exist. Consider all level planar embeddings of H which have the same permutation of level $k + 1$ as $\mathcal{H}$. None of these embeddings has this permutation on level $2k + 1$, too (otherwise G would be cyclic level planar with the same permutation on level 1). Of all these embeddings choose the one which has the minimal permutation on level $2k+1$ in the lexicographical ordering with respect to the ordering of level $k+1$. We

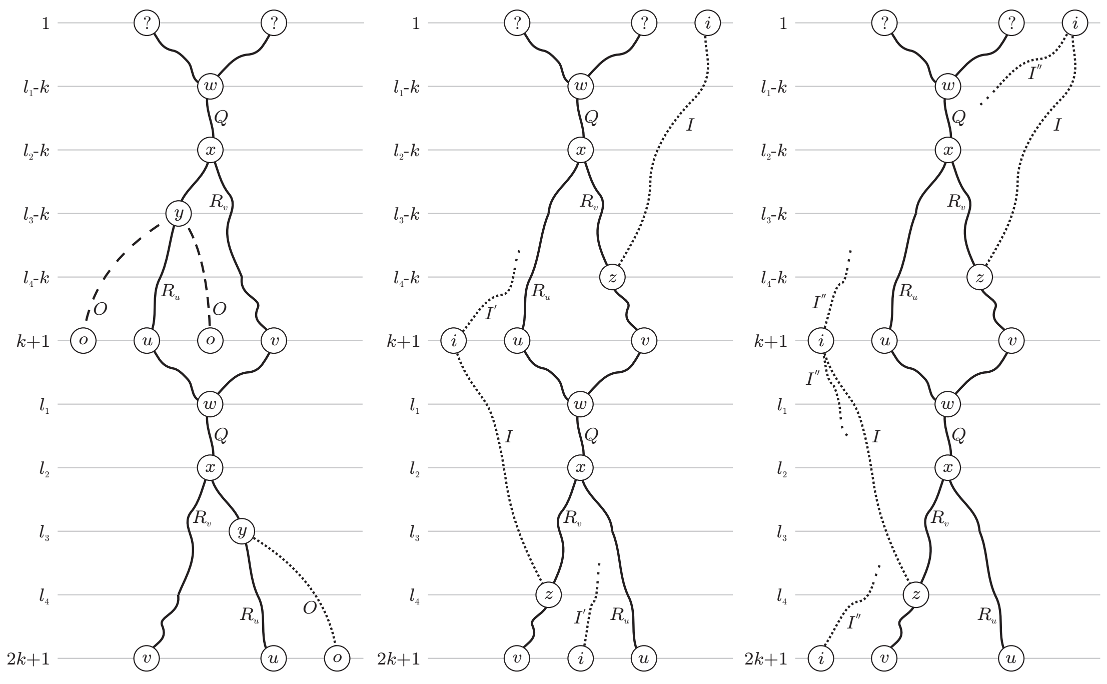

(a) Path O (b) Path I, case 2 (c) Path I, case 3

Figure 6.8. Sketch for the proof of Lemma 6.2

use this embedding from now on. We show that we can construct a new embedding with an even smaller permutation on level $2k+1$, which is a contradiction.

As the permutations of level $k+1$ and $2k+1$ are not the same, there exist two vertices $u, v \in V$ such that $u^{(k+1)} < v^{(k+1)}$ but $u^{(2k+1)} > v^{(2k+1)}$. Let $u^{(2k+1)}$ and $v^{(2k+1)}$ be a pair of such vertices with the maximal number of vertices between them. See Figure 6.8 for a sketch of the various cases in which we omit the indices indicating the levels.

As G is strongly connected but does not match CC, each vertex lies on a 1-cycle. Therefore, there exist paths Q_u from $u^{(k+1)}$ to $u^{(2k+1)}$ and Q_v from $v^{(k+1)}$ to $v^{(2k+1)}$. As the embedding is level planar, Q_u and Q_v cannot be disjoint. Therefore, we have a path Q_l from $u^{(k+1)}$ to $v^{(2k+1)}$ and a path Q_r from $v^{(k+1)}$ to $u^{(2k+1)}$. As G does not match CC, Q_l and Q_r cannot be disjoint. Even more, there has to exist a vertex which lies on all paths from $u^{(k+1)}$ to $v^{(2k+1)}$ and on all paths from $v^{(k+1)}$ to $u^{(2k+1)}$ (otherwise the leftmost path from $u^{(k+1)}$ to $v^{(2k+1)}$ and the rightmost path from $v^{(k+1)}$ to $u^{(2k+1)}$ would be disjoint and generate a graph in CC). Let $w^{(l_1)}$ and $x^{(l_2)}$ be the uppermost and lowest such vertices, respectively, and Q be one path between them.

Let R_v be the leftmost path from $x^{(l_2)}$ to $v^{(2k+1)}$. Let R_u be the rightmost path from $x^{(l_2)}$ to $u^{(2k+1)}$. We now prove that we can flip all vertices between R_v and R_u creating a smaller permutation on level $2k+1$ in the lexicographical ordering and, hence, derive a contradiction. We do so by showing that there are no incoming or outgoing paths left of R_v or right of R_u.

Survey the vertices on the path R_u from level l_2+1 to $2k$. Assume for contradiction that a vertex $y^{(l_3)}$ on the path R_u has an outgoing edge right to R_u starting a path O which we follow downwards (see Figure 6.8(a)). As G is strongly connected, the path O cannot end in a vertex without outgoing edges before it reaches the level $2k+1$. If O reaches R_u again, then R_u was not the rightmost path. Therefore, it has to reach a vertex $o^{(2k+1)}$ on level $2k+1$ right of $u^{(2k+1)}$. Note that the path O between the vertices $y^{(l_3)}$ and $o^{(2k+1)}$ exists a second time in H: Its copy starts at the vertex $y^{(l_3-k)}$ and reaches the vertex $o^{(k+1)}$. As O is disjoint to R_v, $o^{(k+1)}$ has to lie left of $v^{(k+1)}$ (we do not know whether $o^{(k+1)} < u^{(k+1)}$ or $o^{(k+1)} > u^{(k+1)}$ holds). But then $o^{(2k+1)}$ and $v^{(2k+1)}$ have the wrong orientation according to the permutation of level $k+1$ as well and have more vertices between them than $v^{(2k+1)}$ and $u^{(2k+1)}$. A contradiction. The same argument can be used for R_v (switching left and right).

Now consider the vertices on the path R_v from level l_2+1 to $2k+1$. Assume for contradiction that a vertex $z^{(l_4)}$ on the path has an incoming edge left to R_v. We follow this path I upwards (see Figures 6.8(b) and (c)). If I ends on R_v below or on $x^{(l_2)}$, then R_v was not the leftmost path. If I ends on $Q \setminus \{x^{(l_2)}\}$, then $x^{(l_2)}$ would not lie on each path from $u^{(k+1)}$ to $v^{(2k+1)}$. If it ends on the leftmost connection of $u^{(k+1)}$ to $w^{(l_1)}$ above $w^{(l_1)}$, then there would be disjoint paths from $u^{(k+1)}$ to $v^{(2k+1)}$ and from $v^{(k+1)}$ to $u^{(2k+1)}$. The only remaining possibility is that I reaches the level $k+1$ on a vertex $i^{(k+1)}$ left of $u^{(k+1)}$. Again, this path I exists between the vertices $i^{(1)}$ and $z^{(l_4-k)}$ as well. The vertex $i^{(1)}$ has to lie right of $u^{(1)}$ and $v^{(1)}$ due to the path I. However, we do not know whether $u^{(1)} < v^{(1)}$ or $u^{(1)} > v^{(1)}$ holds.

We now examine the position of $i^{(2k+1)}$. If $i^{(2k+1)} > u^{(2k+1)}$ and, therefore, $i^{(2k+1)} > v^{(2k+1)}$, then $i^{(2k+1)}$ and $v^{(2k+1)}$ have the wrong orientation and more vertices between them than $u^{(2k+1)}$ and $v^{(2k+1)}$. If $i^{(2k+1)}$ lies between $v^{(2k+1)}$ and $u^{(2k+1)}$ (see Figure 6.8(b)), we consider the path I' from $i^{(2k+1)}$ upwards. If I' reaches R_u, then we have disjoint paths from $v^{(k+1)}$ to $i^{(2k+1)}$ and from $i^{(k+1)}$ to $v^{(2k+1)}$ and G matches CC. If I' reaches R_v first, then I' would cause a crossing from $i^{(k+1)}$ upwards. The remaining possibility is that $i^{(2k+1)} < v^{(2k+1)} < u^{(2k+1)}$ (see Figure 6.8(c)). Now the same path I'' from $i^{(1)}$ to $i^{(k+1)}$ and from $i^{(k+1)}$ to $i^{(2k+1)}$ has to exist. I'' cannot be disjoint with Q_u or Q_v as $i^{(1)}$ is right of $u^{(1)}$ and $v^{(1)}$ but $i^{(k+1)}$ is left of $u^{(k+1)}$ and $v^{(k+1)}$. Therefore, from $i^{(2k+1)}$ upwards, I'' has to reach I or R_v first in order to reach Q_u. In both cases, a crossing from $i^{(k+1)}$ upwards occurs, which is a contradiction. The same argument can be used to show that no path from R_u upwards exists.

As a consequence, we do not have any outgoing or incoming edges on R_v to the left between l_2+1 and $2k+1$. Analogously, we do not have outgoing or incoming edges to the right of R_u between the same levels. So we can flip the subgraph between R_v and R_u and create a permutation of level $2k+1$ which is smaller than the given one in the lexicographical ordering with respect to the ordering of level $k+1$. A contradiction. Consequently, G is cyclic level planar with an embedding $\mathcal{G}$ such that the permutation of level 1 in $\mathcal{G}$ is the permutation of level $k+1$ in $\mathcal{H}$. $\square$

Figure 6.2 shows a strongly connected cyclic 5-level graph and Figure 6.3 an arbitrary level plane drawing of its $(2, 1)$-hierarchy. Note that levels 1, 6, and 11 have three different permutations. According to Lemma 6.2, we can fix the permutation of level 6 and change the permutation of level 11 to the one of level 6. We search for two vertices on level 11 which have the wrong orientation according to level 6 and the maximal number of vertices between them. These vertices are $u = 1$ and $v = 9$. We get $w = x = 6$ and flip the tree below vertex 6. After that 1 and 2 have the wrong orientation (with $w = 4$ and $x = 6$) and we flip these two vertices. Thereafter, levels 6 and 11 have the same permutation. This permutation is used for the cyclic level plane drawing in Figure 6.2.

Theorem 6.1. *Let G be a strongly connected cyclic k-level graph. G is cyclic level planar if and only if it does not match a pattern in SCLNP.*

Proof. "$\Rightarrow$" We show the contrapositive. If G matches a pattern of SCLNP', then there exists $c \in \mathbb{N}$ s. t. the $(c, 1)$-hierarchy H of G matches an MHLNP pattern. Therefore, H is level non-planar. According to Proposition 6.2, G is cyclic level non-planar then. If G matches the pattern CC, then G is cyclic level non-planar according to Proposition 6.1.

"$\Leftarrow$" We show the contrapositive. Let G be cyclic level non-planar and let H be its $(2, 1)$-hierarchy. If H is level non-planar, then H matches a MHLNP pattern and, therefore, G matches a SCLNP' pattern. If H is level planar, then (the contrapositive of) Lemma 6.2 implies that G matches the pattern CC. $\square$

Note that according to Lemma 6.2, for each strongly connected cyclic k-level non-planar graph not matching CC, its $(2, 1)$-hierarchy matches an MHLNP pattern. Therefore, patterns in SCLNP' can be limited to $2k + 1$ levels.

6.4.2 Cyclic Level Planarity Testing and Embedding

We could use Lemma 6.2 to construct a cyclic level planarity testing and embedding algorithm in the following way: We construct the $(2, 1)$-hierarchy H of the cyclic k-level graph G and permute the level $2k + 1$ until it has the same permutation as level $k + 1$. It that fails then G matches one of the patterns in SCLNP. To achieve linear time complexity, all these searches for the right subgraphs to flip and the flipping itself would have to be made in linear time in total. However, an even simpler algorithm is possible: If the input graph G does not match a pattern in SCLNP then its $(2, 1)$-hierarchy is level planar and we know due to Lemma 6.2 that the permutation of level $k + 1$ can be used in a cyclic level planar embedding. Hence, we construct the $(1, 1)$-hierarchy of G and force it to use this permutation on the first and last level to construct the cyclic level planar embedding. We do so by constructing a frame around the $(1, 1)$-hierarchy which we call the *rigid* $(1, 1)$-*hierarchy*:

Definition 6.8. *Let G be a cyclic k-level graph and H the $(2, 1)$-hierarchy of G. Let H be level planar with embedding $\mathcal{H}$. Let $F = (v_1^{(k+1)}, v_2^{(k+1)}, \ldots, v_s^{(k+1)})$ be the*

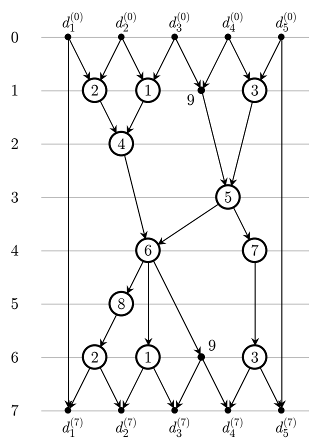


Figure 6.9. Rigid $(1,1)$-hierarchy of the graph G of Figure 6.2

permutation of level $k+1$ in $\mathcal{H}$. The rigid $(1,1)$-hierarchy H' of $\mathcal{H}$ *consists of the $(1,1)$-hierarchy of G and the additional levels 0 and $k+2$. Level 0 has the vertices $d_1^{(0)}, d_2^{(0)}, \ldots, d_{s+1}^{(0)}$ and level $k+2$ the vertices $d_1^{(k+2)}, d_2^{(k+2)}, \ldots, d_{s+1}^{(k+2)}$. H' contains the edges $(d_i^{(0)}, v_i^{(1)})$, $(d_{i+1}^{(0)}, v_i^{(1)})$, $(v_i^{(k+1)}, d_i^{(k+2)})$ and $(v_i^{(k+1)}, d_{i+1}^{(k+2)})$ for each $i \in \{1, \ldots, s\}$ as well as the edges $(d_1^{(0)}, d_1^{(k+2)})$ and $(d_{s+1}^{(0)}, d_{s+1}^{(k+2)})$.*

Figure 6.9 presents the rigid $(1,1)$-hierarchy of the graph G of Figure 6.2. Note that the rigid $(1,1)$-hierarchy H' of $\mathcal{H}$ is level planar if and only if it has an embedding such that the levels 1 and $k+1$ have the same permutation F (or both levels have the permutation F reversed). Then, H is strongly level planar and G cyclic level planar. From Theorem 6.1 we obtain the following idea for a cyclic level planarity testing and embedding algorithm.

Let $G = (V, E, \phi)$ be a strongly connected cyclic k-level graph. We first test whether $|E| \leq 3|V| - 6$ holds (otherwise G cannot be (cyclic level) planar (Euler)). We construct the $(2,1)$-hierarchy H of G then. If H is level non-planar, then G is cyclic level non-planar. Otherwise let $\mathcal{H}$ be a level planar embedding of H. We construct the rigid $(1,1)$-hierarchy H' of $\mathcal{H}$ and test its level planarity. If it fails G is cyclic level non-planar. If it does not fail, we transform the level planar embedding $\mathcal{H}'$ of H' into a cyclic level planar embedding $\mathcal{G}$ of G in a straight forward way: Let $v \in V$ with $\phi(v) \neq 1$. Let $v^{(l)}$ be the corresponding vertex in H'. We set $N^-(v) = N^-(v^{(l)})$ and $N^+(v) = N^+(v^{(l)})$. For a vertex $v \in V$ with $\phi(v) = 1$, we set $N^-(v) = N^-(v^{(k+1)})$ and $N^+(v) = N^+(v^{(1)})$. In both cases, we identify the vertices in H' with the corresponding vertices in G. As a trivial postprocessing, we remove the dummy vertices which were introduced while splitting level 1 and update the adjacency lists accordingly.

Theorem 6.2. *Cyclic level planarity testing and embedding on strongly connected cyclic level graphs can be achieved by Algorithm 6.1 in linear time.*

Algorithm 6.1. cyclicLevelPlanarEmbedding

```
Input: A strongly connected cyclic $k$-level graph $G = (V, E, \phi_G)$
Output: A cyclic level planar embedding $\mathcal{G}$ or $false$
1  if $|E| > 3|V| - 6$ then
2      return $false$
3  Let $H$ be the $(2, 1)$-hierarchy of $G$
4  if ¬levelPlanar($H$) then
5      return $false$                    // $G$ matches a pattern of SCLNP'
6  Let $\mathcal{H}$ be a level planar embedding of $H$
7  Let $H'$ be the rigid $(1, 1)$-hierarchy of $\mathcal{H}$
8  if ¬levelPlanar($H'$) then
9      return $false$                    // $G$ matches the pattern CC
10 Let $\mathcal{H}'$ be a level planar embedding of $H'$
11 Construct cyclic level planar embedding $\mathcal{G}$ of $G$ from $\mathcal{H}'$
12 return $\mathcal{G}$
```

Proof. The correctness of Algorithm 6.1 follows directly from Theorem 6.1. To prove its time complexity, we examine the construction of the $(2, 1)$-hierarchy H of G first. The addition of dummy vertices on level 1 increases the number of vertices and edges by at most $|E|$. Afterwards, the graph is duplicated: Each vertex on level 1 has three duplicates, all remaining vertices and all edges have two duplicates. Therefore, the size of H is linear in the size of G. All steps can easily be done in linear time. To test the level planarity of H, any linear time level planarity testing and embedding algorithm for hierarchies can be used as a black box [33, 86]. The construction of the $(1, 1)$-hierarchy is possible in linear time as well. Let w be the number of vertices on level 1 in this hierarchy. To build the rigid $(1, 1)$-hierarchy, we add $2(w + 1)$ vertices and $4w + 2$ edges and again use a linear time level planarity testing and embedding algorithm. The construction of $\mathcal{G}$ from $\mathcal{H}'$ can again easily be done in linear time. $\square$

6.5 Summary

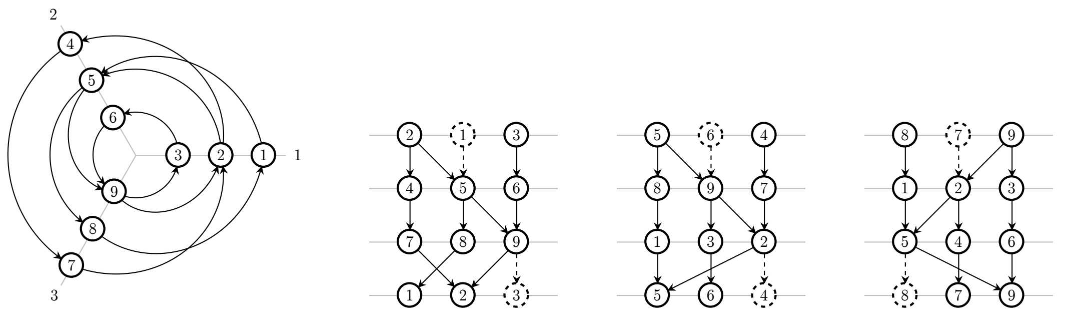
We presented the idea to use level non-planarity patterns to prove cyclic level non-planarity and have shown that each of the seven minimal level non-planarity patterns for hierarchies (MHLNP) of Healy et al. [78] are necessary in the strongly connected cyclic level case. To build a complete set, an eighth pattern is needed: a simple cycle wrapping around the center more than once (CC).

This characterization leads to a simple linear time cyclic level planarity testing and embedding algorithm which can use any (linear time) level planarity testing and embedding algorithm as a black box.

6.6 Open Problems

In this chapter, we defined SCLNP patterns as level non-planarity patterns wrapping around the center at most twice. Hence, a pattern can overlap with itself. As an open problems remains to describe the structure of the SCLNP patterns in a more direct way. An important step towards that would be the proof of the following conjecture:

Conjecture 6.1. *Let G be a strongly connected cyclic k-level graph. G is cyclic level planar if and only if G does not match a pattern in SCLNP with the patterns in SCLNP' using at most $k+1$ levels.*



(a) Graph G (b) $(1,1)$-hierarchy (c) $(1,2)$-hierarchy (d) $(1,3)$-hierarchy

Figure 6.10. Different hierarchies of a cyclic level non-planar graph G

Put differently, we conjecture that the patterns in SCLNP' to not overlap at all, that the upper and lower end of the patterns touch at most, and, hence, use $k+1$ levels at most. Consider the cyclic level non-planar graph G with $k=3$ levels in Figure 6.10(a) as an example of a graph where $k+1=4$ levels suffice. There are three possibilities to cut the graph open which result in its $(1,1)$-hierarchy (see Figure 6.10(b)), its $(1,2)$-hierarchy (see Figure 6.10(c)) and its $(1,3)$-hierarchy (see Figure 6.10(d)). All these hierarchies contain the level non-planarity pattern $C2$ (a cycle with two emerging paths). The vertices and edges not needed for the pattern are drawn dashed. This is only an example and does not prove that $k+1$ level suffice in general, of course. However, it does prove that k level do not suffice. Note that all three hierarchies are drawn such that their only crossing is between the level $k=3$ and $k+1=4$. On the other hand, Lemma 6.3 is a strong indication for Conjecture 6.1:

Lemma 6.3. *Let G be a strongly connected cyclic k-level graph not matching CC. Let H be its $(2,1)$-hierarchy. If H matches an MHLNP pattern P using more than $k+2$ levels, then it matches another instance of an MHLNP pattern.*

Proof. W.l.o.g. let 1 be the first level of the pattern P in H. Note that H matches the pattern P, but the $(2,2)$-hierarchy H' of G does not match it as level 1 of G is missing at the top. As more than $k+2$ levels are used by the pattern P, the pattern cannot occur from level k downwards completely as well. H' has to match another MHLNP pattern as we could use H' instead of H in Lemma 6.2 without change of the proof. □

Note that Lemma 6.3 does not make a statement on patterns using exactly $k+2$ levels. Nevertheless, we conjecture that the patterns use $k+1$ levels at most as all strongly connected cyclic level graphs matching a longer pattern seem to match a shorter pattern or the CC pattern as well.

A proof of Conjecture 6.1 would be an important step to prove the minimality of the SCLNP patterns. Although we use minimal patterns for hierarchies we could not conclude that the SCLNP patterns are minimal as well as the potential overlapping of the patterns leads to new implications. The new pattern CC is minimal, however, as removing an edge results in a simple path.

Another open problem is to find a (linear time) cyclic level planarity testing and embedding algorithm for arbitrary cyclic level graphs. Finding the cyclic level non-planarity patterns for this case is of interest as well. The minimal level non-planarity patterns for arbitrary level graphs have already been extended twice [53, 61]. This shows that the patterns for arbitrary (cyclic) level graphs are rather complicated. Regarding an algorithm we make the following conjecture:

Conjecture 6.2. *Let G be a (not necessarily strongly connected) cyclic k-level graph. G is cyclic level planar if and only if there does not exist $c \in \mathbb{N}$ s.t. the $(c,1)$-hierarchy is level non-planar and G does not contain an undirected simple cycle wrapping around the center more than once.*

Testing $(2,1)$-hierarchies only does not suffice for arbitrary cyclic level graphs. Consider a connected proper level non-planar graph $G = (V, E, \phi)$ with l levels which consists of a MHLNP pattern only. Let $G' = (V, E, \phi')$ be the cyclic 3-level graph with $\phi'(v) = ((\phi(v) - 1) \mod 3) + 1$. Informally, G' is a version of G which is wrapped around the center $\lceil \frac{l}{3} \rceil$ times. It is still a DAG and hence not strongly connected. Furthermore, it is cyclic level non-planar as G was level non-planar. However, the $(2,1)$-hierarchy of G' consists of many connected components as each of them represents two windings of G' only. Each of these components is level planar as in each of them some edges of the MHLNP pattern are missing. To have at least one complete and connected version of G' if a hierarchy, the $(\lceil \frac{l}{3} \rceil, 1)$-hierarchy is needed. In general, a $(c,1)$-hierarchy with $c \in \Theta(|V|)$ is needed for a non-proper cyclic level graph $G = (V, E, \phi)$. This hierarchy, however, is already quadratic in the size of the input graph G.

Another approach to prove a correct cyclic level planarity testing and embedding algorithm is to adapt the algorithms by Randerath et al. [124] or Healy et al. [76, 77]. However, in the proofs in these papers details necessary for the transformation to cyclic level planarity are omitted. Up to now, we were not able to get a complete version of the proofs by contacting the authors or find one ourselves.

A final open problem concerns straight-line drawings, i. e., straight-line intermediate drawings or cyclic drawings using spirals without bends for long edges. In the hierarchical case any planar level graph is drawable without edge bends [42]. However, this can lead up to exponential width [35]. This lower bound on the width holds in the cyclic case trivially, as any k-level graph is a cyclic k-level graph as well. However, the lower bound could be higher in the cyclic case. Furthermore, to the best of our knowledge, the problem of whether any cyclic level planar graph is drawable without bends is open.

7 Comparison of the Hierarchical and Cyclic Sugiyama Framework

In Chapter 1.3.1 we stated several drawbacks of the hierarchical Sugiyama framework, e. g., potentially longer edges and more crossings. In this chapter we compare the resulting drawings of the complete hierarchical and cyclic Sugiyama frameworks empirically. We state the question whether there are classes of graphs where the drawings really benefit from the cyclic drawing style.

Essentially, there are four questions to settle first. Which algorithms do we choose for each phase? Which parameters do we use for each algorithm? Which graphs do we test? And finally, which properties of the drawings do we measure? Obviously, there is an unmanageable amount of possible combinations here. However, we do not want to test the different phases, but compare the two drawing styles themselves. Hence, we restrict ourselves to the following:

Regarding the algorithms to choose, we test only one combination for the hierarchical and one for the cyclic framework. We use the best implementations available in the graph visualization toolkit Gravisto which are all reasonable heuristics. For the decycling in the hierarchical case we use a refinement of the greedy cycle removal algorithm [47], which breaks the strongly connected components by temporarily removing incoming edges of vertices with high out-degree. For the leveling we use the algorithm by Coffman and Graham [29] in the hierarchical case and our best cyclic leveling algorithm, i. e., the spring embedder heuristic with MST initialization (see Chapter 3.3.3). We use global sifting as a crossing reduction in both cases (see Chapter 4.4). The algorithm by Brandes and Köpf is used in its original [19] and cyclic variant (see Chapter 5.3) for the coordinate assignment.

Regarding the parameters of these algorithms, we choose the following values. The decycling algorithm in the horizontal case does not have any parameters. Regarding the leveling we chose $\lceil\sqrt{2|V|}\rceil$ as the width for both frameworks. In the cyclic case we chose $\lceil\sqrt{0.8|V|}\rceil$ as the number of levels as this seems to give the best results. This is supported by preliminary tests not shown in this thesis. We perform ten rounds of global sifting in both frameworks. We execute all four runs in the cyclic coordinate assignment algorithm using maximal blocks. Thereby, we can measure if shearing the drawing really leads to quadratic width in practice.

We constructed graphs from 100 to 500 vertices in steps of ten with $|E| = 1.5|V|$. The main question to settle is the structure of the graphs to construct. We used the following graph generator which uses one additional parameter l: Let $V = \{v_1, \ldots, v_m\}$ be the set of m vertices to construct. We assume V to be cyclically ordered. To add an edge, we chose two vertices $v_i \neq v_j$ randomly. We add the edge (v_i, v_j) only if v_j is one of the l vertices after v_i in the cyclic order. More formally, this is the case if $j > i$ and $j - i \leq l$ or if $j < i$ and $j - i + m \leq l$. Choosing $l = m$, i.e., $l = |V|$ gives a completely random graph without any cyclic structure. When choosing $l = 1$, only edges from each vertex to its direct successor are allowed. Therefore, the result is a subgraph of a cycle. Choosing values of l between 1 and $|V|$ regulates the cyclic structure of the graph. We used the values $l = 0.5|V|$ and $l = 0.1|V|$ in our benchmarks. Using the value $l = 0.5|V|$ in the graph generator corresponds to choosing two vertices v_i and v_j repeatedly and adding either the edge (v_i, v_j) or (v_j, v_i) depending on which edge is shorter in the cyclic order. However, the resulting graphs still do not have a clear cyclic structure. Using $l = 0.1|V|$ gives the desired cyclic structure as vertices are only connected to the next 10% of vertices.

Each of these graphs consists essentially of one long cycle with some shortcuts. Another approach is to construct graphs with many interweaved shorter cycles. However, this would result in a larger feedback arc set. This would be a huge advantage for the cyclic Sugiyama framework as there no edges have to be reversed. To keep the benchmark more balanced we choose the graph generator as described above. Obviously, all these decisions are debatable but we had to restrict ourselves to keep the number of diagrams reasonable.

We measured the following properties of the drawings: percentage of reversed edges, number of levels, span of the edges, number of crossings, number of bends per edge, width of the drawing, and the displacement of the outer and inner segments, i.e., the difference in the x-coordinates of the end vertices of the segments. We present two diagrams for each of these properties, i.e., one for the parameter $l = 0.5|V|$ and one for $l = 0.1|V|$. To avoid confusion, we do so even if the hierarchical or cyclic framework does not produce data for the current property and the diagram contains only one line. In all these diagrams the absolute values are less important than the comparison of the hierarchical and cyclic framework.

Additionally we measured the running time of both frameworks. We present the results in Figure 7.1 and Figure 7.2 for the sake of completeness only. The results highly depend on the choice of the algorithms for each phase. Furthermore, these algorithms were not implemented by the same person and the cyclic algorithms

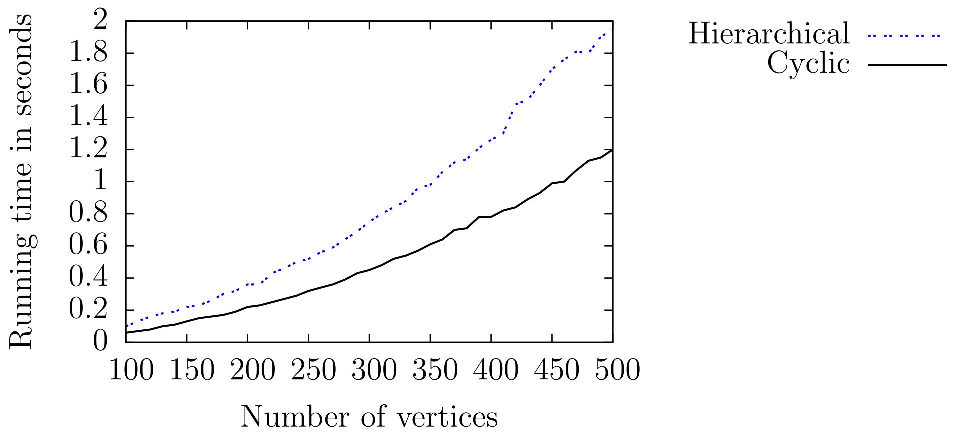


Figure 7.1. Benchmark: running time ($l = 0.5|V|$)

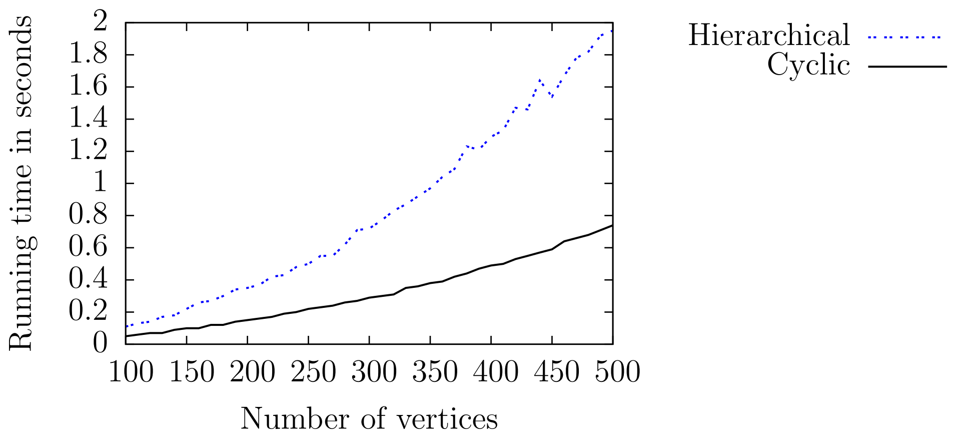


Figure 7.2. Benchmark: running time ($l = 0.1|V|$)

were optimized more than the hierarchical ones. Furthermore, the cyclic Sugiyama framework does not need the decycling phase. At least these diagrams show that the running times of the cyclic algorithms are within the same magnitude of the hierarchical versions.

Figure 7.3 and Figure 7.4 present the percentage of reversed edges. Obviously, this number is zero in the cyclic case. In the hierarchical drawing style, about 4% of all edges are reversed in the case $l = 0.5|V|$ and 2.5% in the case $l = 0.1|V|$. This rather small number of reversed edges was a goal of the choice of the input graphs and gives the possibility to compare the hierarchical and cyclic frameworks in a fair way.

The number of levels is shown in Figure 7.5 and Figure 7.6. This number is given as an input in the cyclic case and is chosen to achieve an even distribution of the original vertices on the levels. In the hierarchical case with $l = 0.5|V|$ about twice the number of levels are needed. Consequently, the average span of the edges in

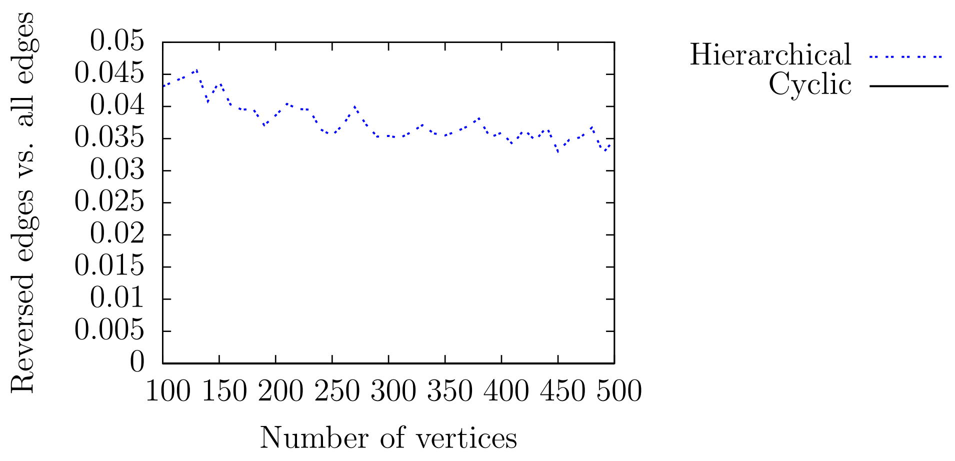


Figure 7.3. Benchmark: fraction of reversed edges ($l = 0.5|V|$)

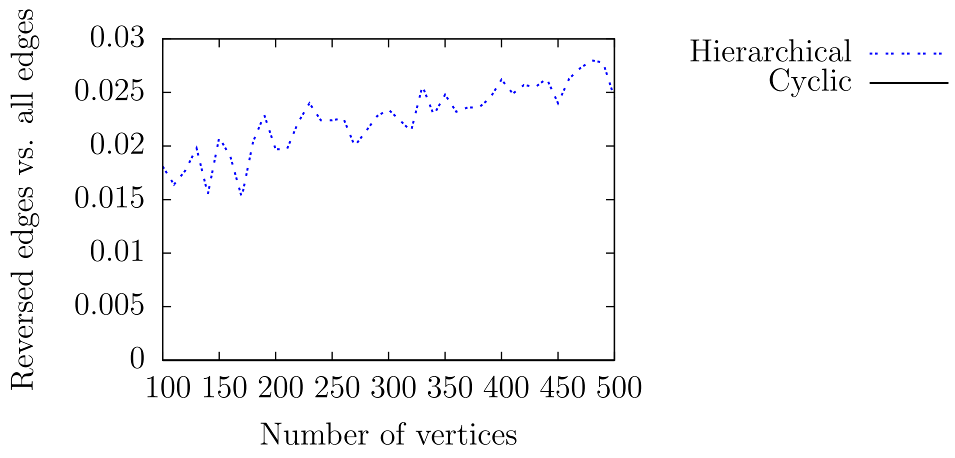


Figure 7.4. Benchmark: fraction of reversed edges ($l = 0.1|V|$)

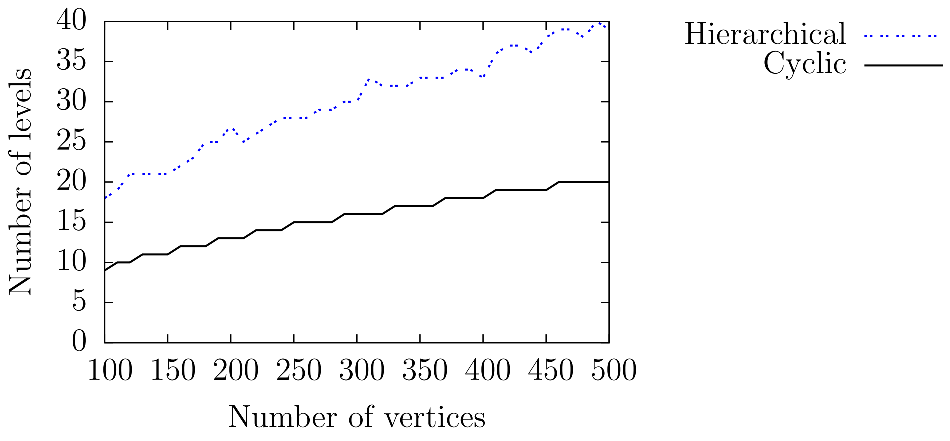


Figure 7.5. Benchmark: number of levels ($l = 0.5|V|$)

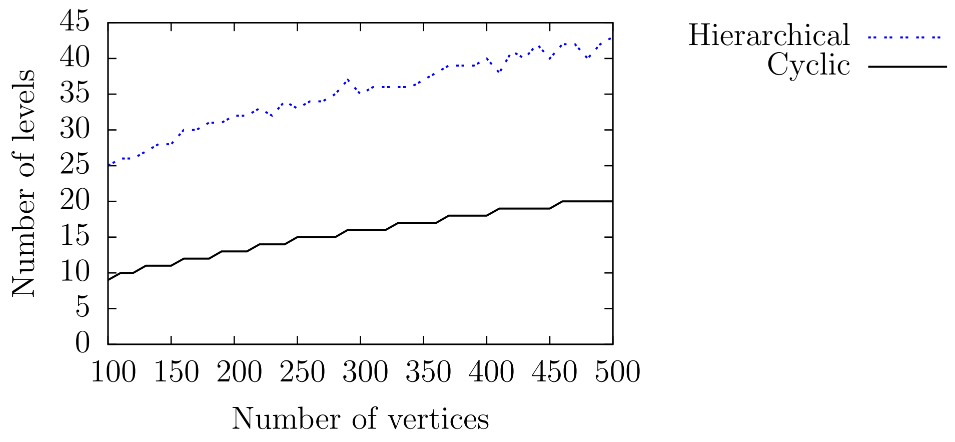


Figure 7.6. Benchmark: number of levels ($l = 0.1|V|$)

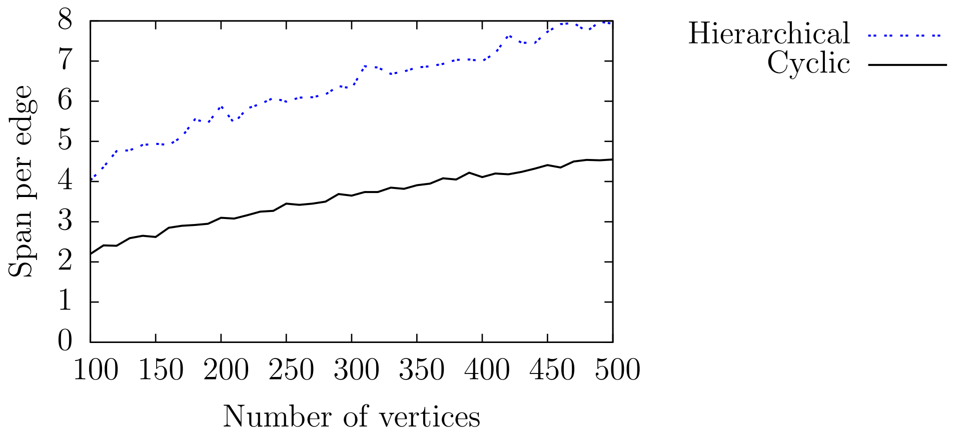


Figure 7.7. Benchmark: average span per edge ($l = 0.5|V|$)

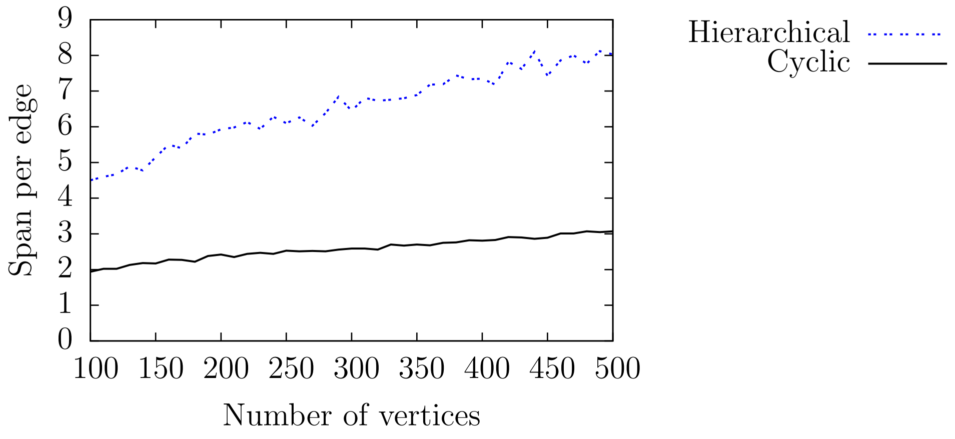


Figure 7.8. Benchmark: average span per edge ($l = 0.1|V|$)

the hierarchical case is nearly twice as high as in the cyclic case as can be seen in Figure 7.7. For $l = 0.1|V|$ the difference in the number of levels and the span (see Figure 7.8) is even higher.

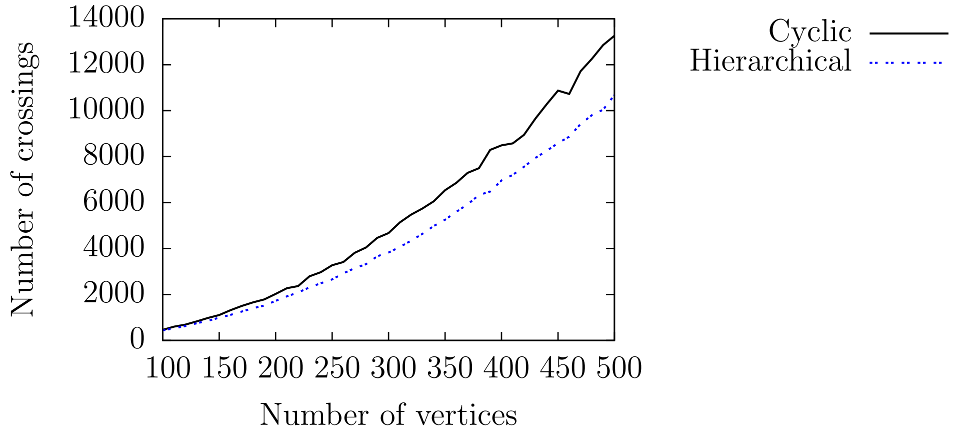


Figure 7.9. Benchmark: number of crossings ($l = 0.5|V|$)

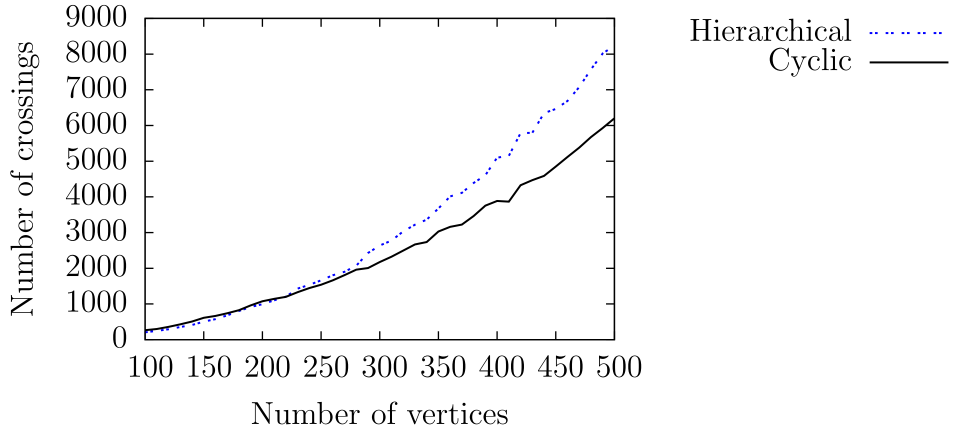


Figure 7.10. Benchmark: number of crossings ($l = 0.1|V|$)

Figure 7.9 and Figure 7.10 present the number of crossings of the resulting drawings. While in the case $l = 0.5|V|$ the number of crossings is about 20% higher in the cyclic case, it is about 20% lower in the cyclic case with $l = 0.1|V|$.

Regarding the number of bends, both coordinate assignment phases guarantee to yield at most two bends per edge. The hierarchical drawings have about 0.9 bends per edge in average. In the cyclic drawings the number of bends is slightly smaller in the case $l = 0.5|V|$ (see Figure 7.11). With $l = 0.1|V|$ (see Figure 7.12) only 0.5 bends per edge are needed, however. Obviously, having shorter edges and the ability to shear inner segments in the cyclic case benefits the number of bends.

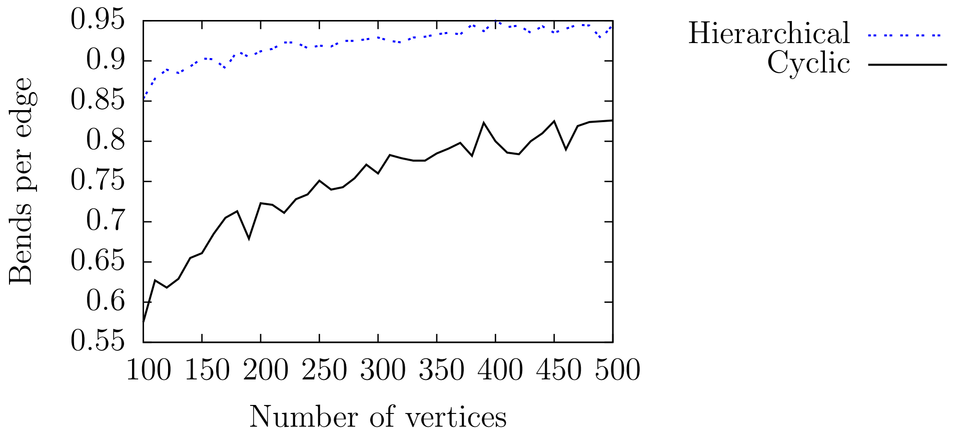


Figure 7.11. Benchmark: number of bends per edge ($l = 0.5|V|$)

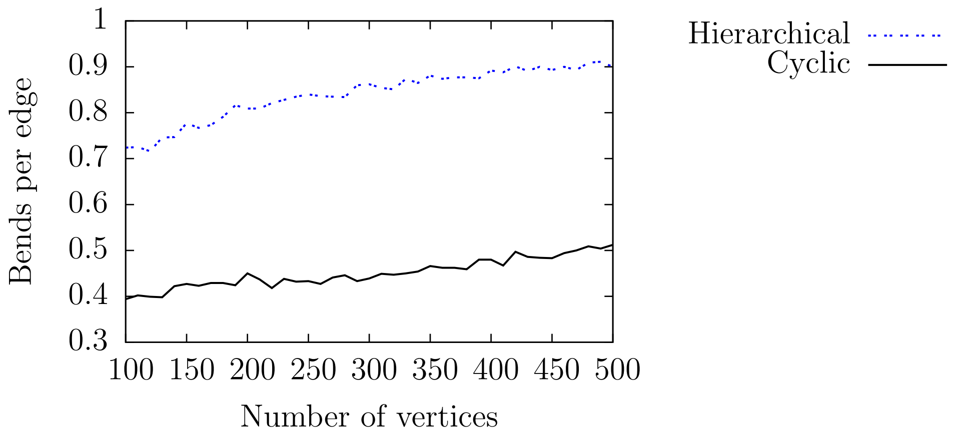


Figure 7.12. Benchmark: number of bends per edge ($l = 0.1|V|$)

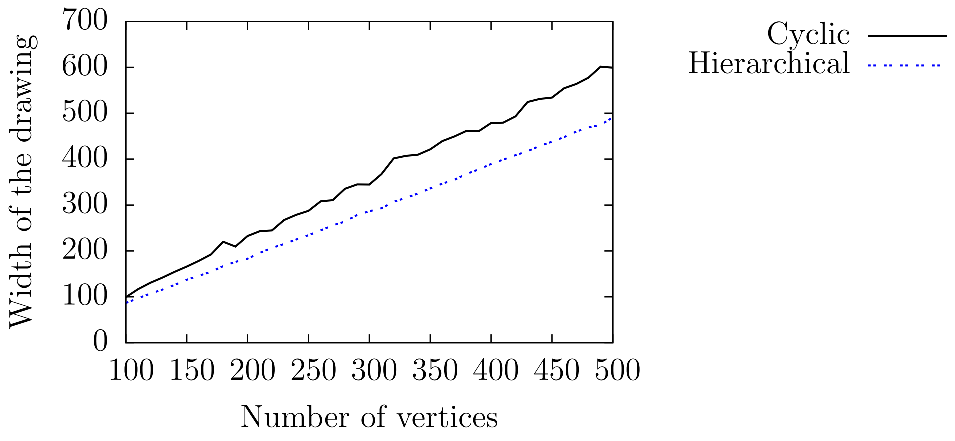


Figure 7.13. Benchmark: width of the drawing ($l = 0.5|V|$)

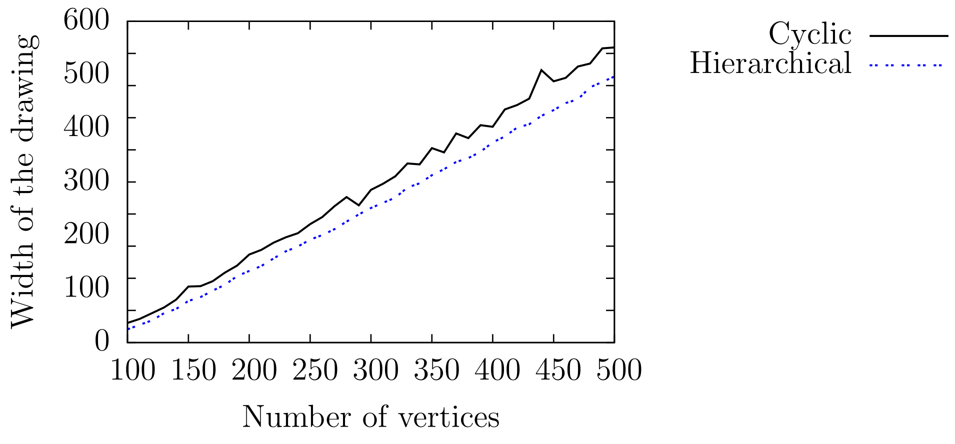


Figure 7.14. Benchmark: width of the drawing ($l = 0.1|V|$)

The width of drawings of the algorithm by Brandes and Köpf is linear in the size of the (not necessarily proper) input graph in the horizontal case. This is reflected in the diagrams in Figure 7.13 and Figure 7.14. Note that the drawings are produced using unit distance. In the cyclic case the theoretical worst case width is quadratic, however. The benchmarks indicate that in most cases the width is still linear and only slightly higher (20% for $l = 0.5|V|$, 10% for $l = 0.1|V|$) than in the hierarchical case.

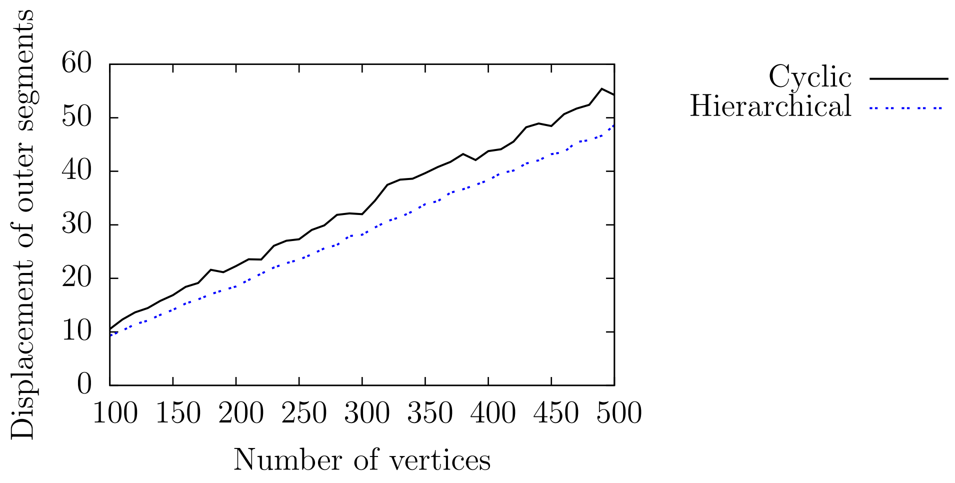


Figure 7.15. Benchmark: average displacement of outer segments ($l = 0.5|V|$)

The last diagrams show the displacements of outer and inner segments, i. e., the average difference of the x-coordinates of the end vertices of these segments. The displacements of the outer segments as depicted in Figure 7.15 and Figure 7.16 are slightly higher in the cyclic case. However, as the number of outer segments in the cyclic case is smaller the sum of the displacements is nearly identical. Comparing the average displacement of an outer segment with the width of the drawing shows

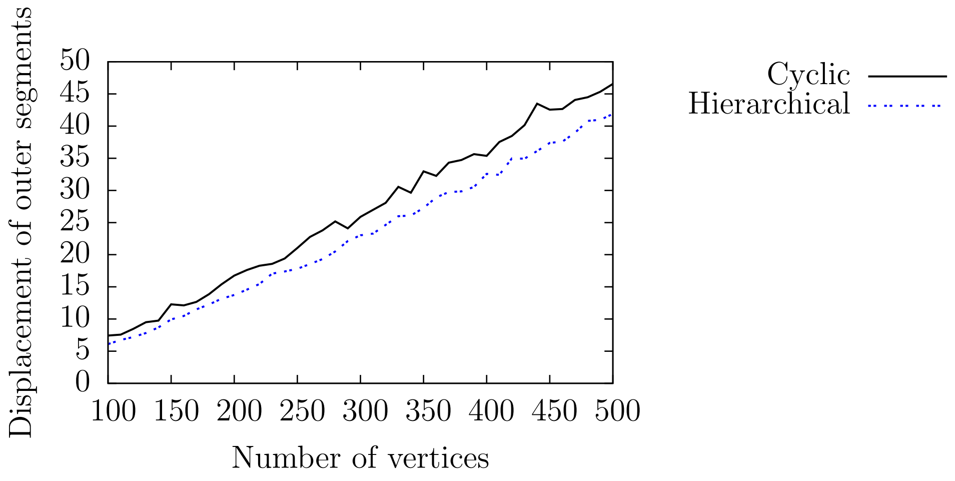


Figure 7.16. Benchmark: average displacement of outer segments ($l = 0.1|V|$)

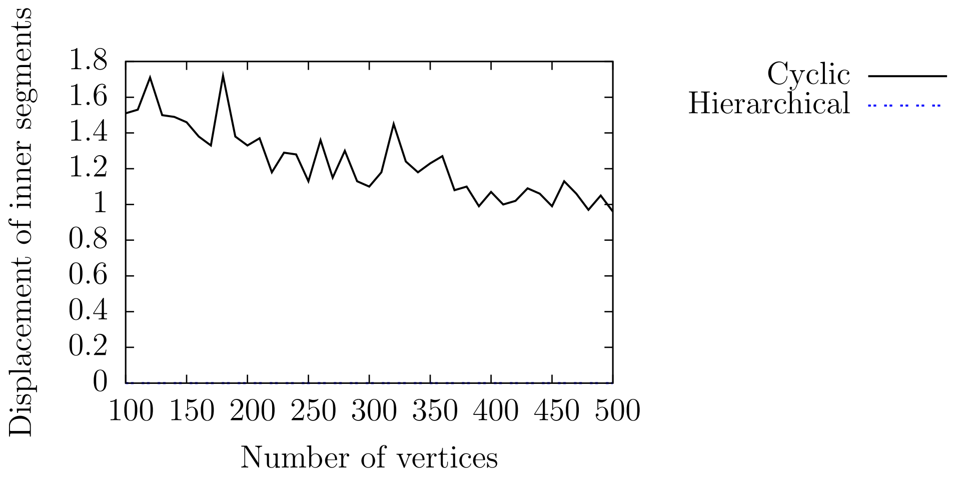


Figure 7.17. Benchmark: average displacement of inner segments ($l = 0.5|V|$)

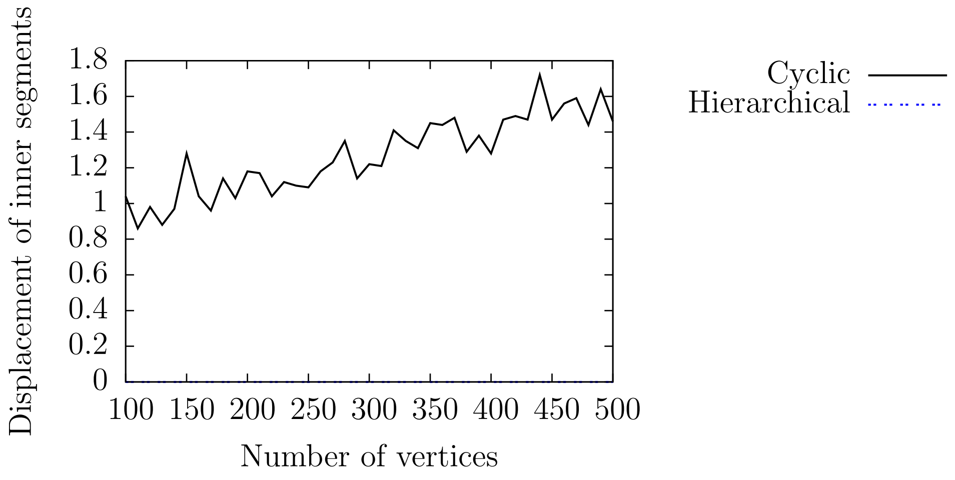


Figure 7.18. Benchmark: average displacement of inner segments ($l = 0.1|V|$)

that in both frameworks an average outer segment has an displacement of about 10% of the width of the drawing.

In the hierarchical case, the inner segments are aligned vertically. Hence, their displacement is zero (see Figure 7.17 and Figure 7.18). In the cyclic case, blocks are sheared. Hence, inner segments have a displacement as well. However, their displacement is much smaller than the displacement of the outer segments. The values are between 1 and 1.5. Note that a displacement of 1 conforms to a slope of 45° in the intermediate drawing which corresponds to the drawings in Figure 5.13 and Figure 5.15.

To summarize the results, the cyclic framework produces drawings with a smaller number of bends, a similar width, and a slightly higher displacement of outer segments to the hierarchical framework. Giving about half the levels needed in the hierarchical setting is reasonable as more levels are not used by the leveling algorithms in the cyclic case. Using this number of levels results in a smaller span in the cyclic scenario. The number of crossings in the cyclic case is slightly larger than in the hierarchical case for $l = 0.5|V|$ and slightly smaller for $l = 0.1|V|$. The large conceptual differences between the two frameworks are the reversed edges in the hierarchical case and the sheared inner segments in the cyclic case.

To summarize in a different way, for $l = 0.1|V|$ the cyclic drawings have no reversed edges, less levels, less span, less crossings, less bends, a slightly higher width and displacement of outer segments at the cost of sheared inner segments. The same holds for $l = 0.5|V|$ except here the cyclic framework produces slightly more crossings than the hierarchical one.

These benchmarks show that there are classes of graphs which benefit from the cyclic drawing style which was the main question to settle in this chapter. One important fact to mention here is that 30 years of research went into the hierarchical Sugiyama framework. The cyclic algorithms for each phase are obviously influenced by these results. However, they are still a first approach only. This is especially true for the leveling algorithms which do not have any similarity with popular hierarchical leveling heuristics. Hence, further research is likely to enlarge the advantage of the cyclic drawing style for special classes of graphs.

8 Conclusion

In this thesis, we presented the extension of the Sugiyama framework to cyclic drawings. One major drawback of the hierarchical framework is the destruction of all cycles in its first phase by reversing some edges. Thus, said cycles are difficult to detect in the drawings. Sugiyama et al. [140] proposed an alternative to such drawings called recurrent hierarchies. A planar recurrent hierarchy is shown on the cover of the book by Kaufmann and Wagner [93]. There it is stated that recurrent hierarchies are "unfortunately [. . .] still not well studied". This does not hold anymore as we investigated the needed leveling, crossing reduction, and coordinate assignment algorithms. The decycling phase is unnecessary, as it is the explicit goal to visualize the cycles as such.

For the leveling phase (see Chapter 3), completely new algorithms were needed as established ones, like the heuristic by Coffman and Graham [29], heavily use the presence of sources and sinks in the graph. We proposed three algorithms. The breadth first search heuristic is simple but gives poor results. The heuristics based on the minimal spanning tree algorithm by Prim [30, 120] and spring embedder [40, 64] perform notably better. The best results were achieved by using the MST heuristic as an initialization of the spring embedder leveling.

Similarly, for the crossing reduction (see Chapter 4), new algorithms had to be found. Existing methods perform a level-by-level sweep which push crossings iteratively to the next level until they are resolved at the last or first level. As a cyclic level graph does not have a last or first level, this resolution would never happen. We proposed a global approach using blocks and giving each block a distinct x-coordinate while permuting them all at the same time. The resulting global barycenter and median performed poorly, but the global sifting algorithm gave good results. When using it for hierarchical crossing reduction, it avoided 20% more crossings than other heuristics which guarantee the absence of type 2 conflicts as well.

The standard algorithm for the hierarchical coordinate assignment phase by Brandes and Köpf [19] was the basis for our cyclic coordinate assignment algorithm (see Chapter 5). However, a new problem arose as cyclic dependencies in the left-to-right order of vertices on different levels prevent all blocks from being vertically aligned. We solved this by shearing such blocks. It is possible to avoid this problem, however, if our global sifting heuristic is used for the crossing reduction and the block building in the coordinate assignment is adapted.

A prototypical implementation of the leveling, crossing reduction, and coordinate assignment algorithms has been integrated in the graph visualization toolkit Gravisto [7]. The current development snapshot can be downloaded by following the instructions on `http://gravisto.fim.uni-passau.de/`.

We also presented a linear time cyclic level planarity testing and embedding algorithm for strongly connected graphs (see Chapter 6). It transforms the input graph to a level graph and can use any linear time level planarity testing and embedding algorithm for hierarchies [33, 86] as a black box.

8.1 Open Problems

Although we gave algorithms for each phase of the cyclic Sugiyama framework, there is space for many refinements and several open problems remain. The following list shows what future research could tackle. Most of these problems have already been solved or have at least been stated with regarding the hierarchical Sugiyama framework. Please note that possible improvements of the presented algorithms were already discussed in the respective chapters.

8.1.1 Dummy Vertices

Dummy vertices are inserted in the graph in the Sugiyama framework to allow a level-by-level sweep in the crossing reduction phase. As our global crossing reduction combines the inner segments of a long edge to a block and sifts the whole block, dummy vertices are not needed in this case. Hence, the question arises if the dummy vertices are not needed at all. This is also of interest in the hierarchical Sugiyama framework as the global crossing reduction can be used there as well.

Dummy vertices are introduced after the leveling phase. The first and second phases are not affected by this change. In the third phase, we use our global crossing reduction which has a time complexity which is independent of dummy vertices. With slight modifications, the algorithm would work without dummy vertices at all. The original coordinate assignment algorithm by Brandes and Köpf [19] as well as our extension to cyclic level graphs builds even larger blocks than the crossing reduction as possibly several edges are combined to one block. Again variants of the algorithms without dummy vertices are possible.

8.1.2 Vertices with Arbitrary Size

Most graph drawing algorithms assume that all vertices are drawn as points. But to understand what the graph represents, in most cases each vertex has to be drawn as a small labeled rectangle or circle. If all vertices are of nearly the same size, algorithms using points for vertices can be used by choosing the minimal distance between the vertices large enough.

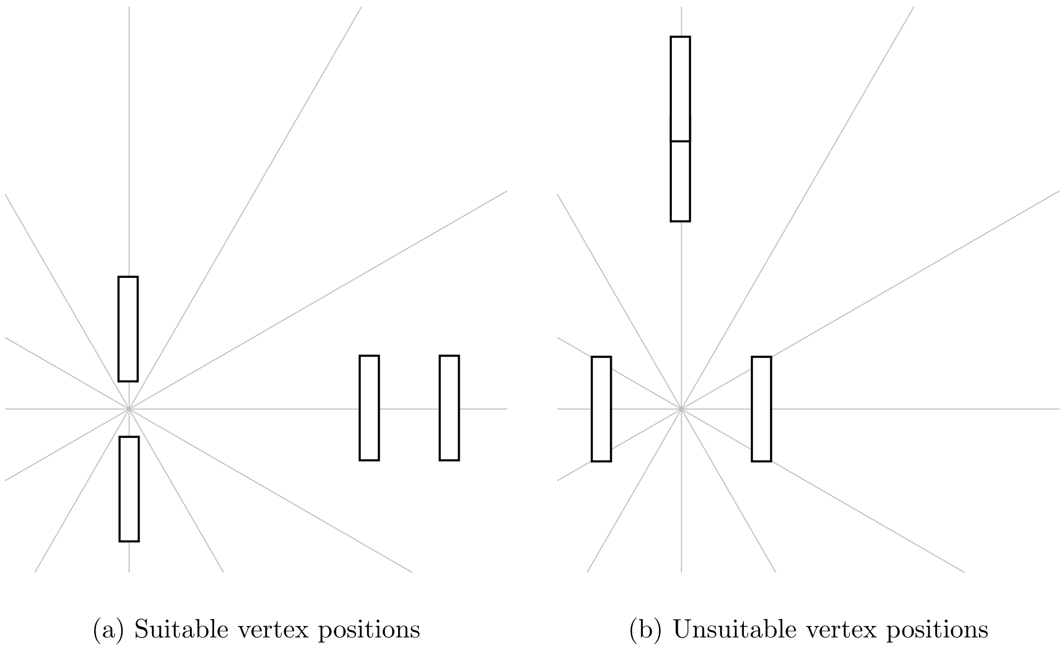

(a) Suitable vertex positions (b) Unsuitable vertex positions

Figure 8.1. Positions of vertices with arbitrary size

However, if the vertices vary in size, much area is wasted when determining the minimum distance by the largest vertex. Here, vertices with individual sizes are needed [63, 132]. In the leveling phase, the width of each vertex has to be taken into account when computing the width of one level. An option in leveling phase is to place high vertices on several levels. If high vertices span more than one level, the crossing reduction phase has to take that into account to avoid crossings between edges and large vertices. Our global sifting crossing reduction could help here by treating a high vertex as one block. The coordinate assignment phase has to deal with the large vertices as well. Here, in some cases, additional bends may be necessary.

When drawing cyclic level graphs on the plane, another new problem arises: Vertices with large aspect ratios are better suited for some places than others, see Figure 8.1. A narrow and high vertex that is placed near the center should be on a level above or below the center. Otherwise it would cross several levels. If several such vertices are placed on the same level far away from the center, they should be placed on levels left or right of the center to avoid overlapping each other. Note that this problem does not arise when using the 3D drawing on a cylinder. Thus, the 2D and 3D drawing styles are not equivalent any more when dealing with vertices with arbitrary sizes.

8.1.3 Toroidal Level Drawings

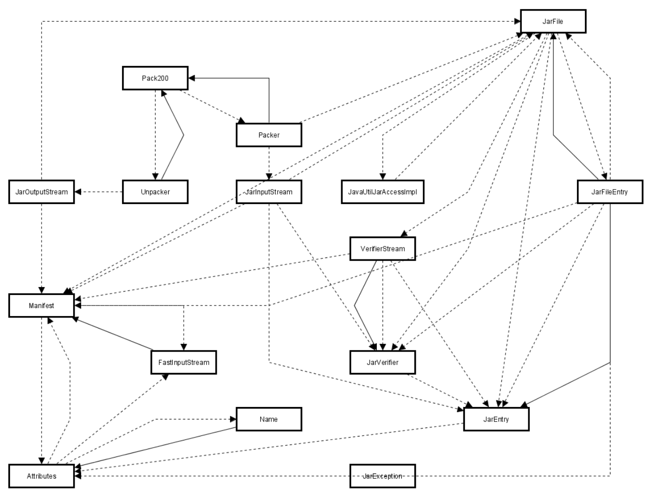
Radial level drawings [2, 6] are a related approach to extend level drawings like cyclic level drawings. Here, the levels form concentric circles and each edge points outwards. See Figure 1.16(b) for an example. In 3D, this can be seen as a drawing on an upright cylinder where the levels are horizontal circles on its surface and all edges point downwards. Cutting the surface of the cylinder open along a vertical line leads to a level drawing where edges can leave the drawing on the left or right hand side and enter on the opposing side. Analogously, cutting the cylinder of a cyclic level drawing along a level line leads to a level drawing where edges leave the drawing at the bottom and enter at the top. If we were to allow reversed edges as well, those would leave at the top and enter at the bottom. Combining both radial and cyclic drawings leads the the possibility of edges leaving at each side of the drawing and entering at the opposing side. In 3D this corresponds to drawings on a torus which we call toroidal level drawings.

Drawing in 3D has few applications in practice and drawing on a torus might have even less. But the 2D drawing in a rectangle in which the edges can leave the rectangle at each side gives an interesting application: Consider the problem of viewing large graphs on a computer screen. As the graph will not fit on the screen at once, only a part of it is visible. To display the rest, the view has to be scrolled. Some vertices of the graph are placed in the center of the drawing and can use the complete 360° to spread its edges whereas vertices in the corner of the drawing can only use 90° for their edges. Using a toroidal drawing, each position in the drawing has the same quality as each position can use the full 360°. Edges can be shorter and cause less crossings. See Figure 8.2 for an example of such drawings. As scrolling is needed anyway to view the large graph, the view can be scrolled infinitely in any direction without reaching the border of the drawing.

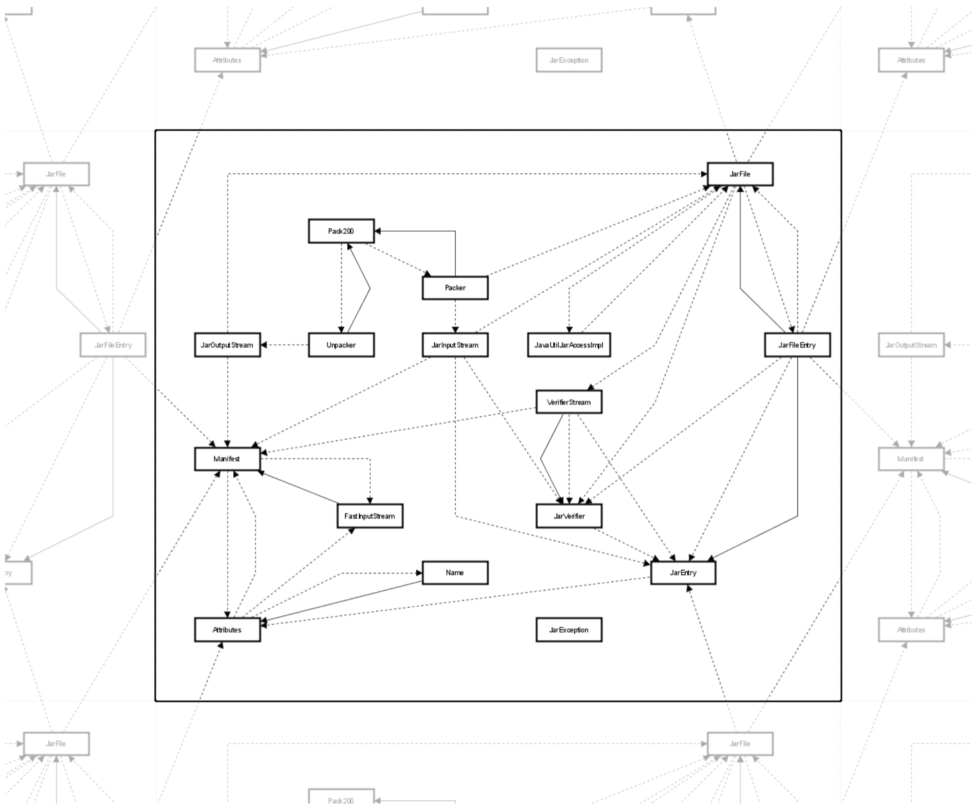
8.1.4 Other Cyclic or Toroidal Drawing Styles

Extending other drawing styles to cyclic or toroidal drawings raises completely new challenges. Extending orthogonal drawings [15, 58, 142] to the cylinder or torus would have applications in the VLSI design or in drawing UML diagrams.

Another possibility are spring embedder [40, 64]. Here, similarly to the leveling problem, the size of the cylinder or torus has to be determined before applying an algorithm. If the area is chosen too large, it can happen that the resulting drawing uses just one part of the area without utilizing the cyclic structure of the whole area. Thus, the area and the repelling and attracting forces should be chosen such that the graph covers the entire area. For the routing of edges, two possibilities seem reasonable: The first is that edges get longer as vertices move apart even when the



(a) In the plane



(b) On a torus

Figure 8.2. UML Diagram of the Java Package `java.util.jar`

vertices come closer again at the opposing side of the torus. The second possibility is that edges are always routed such that they use the shortest possible path. This implies that they can change to a completely different routing when incident vertices move only by a small amount, however.

8.1.5 Clustered Cyclic Level Graphs

In a clustered level graph, special subgraphs, called clusters, are to be drawn in a contiguous region, e. g., a rectangle, see Figure 8.3. Clusters can have a hierarchical structure. These clusters impose some restrictions on the leveling and crossing reduction phases as they have to ensure that the vertices and edges of each cluster are placed on contiguous levels and next to each other on the same level. In the case of horizontal level lines, several algorithms have been proposed [43, 60, 138].

Our global sifting algorithm can easily be applied to the clustered crossing reduction problem in the hierarchical and cyclic case as described in Section 4.6.4. However, for the remaining leveling and coordinate assignment phases for clustered cyclic level graphs, new algorithms are needed.

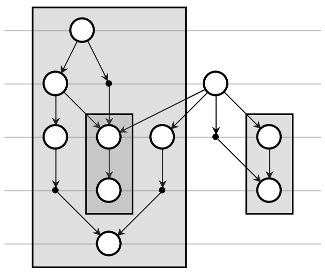


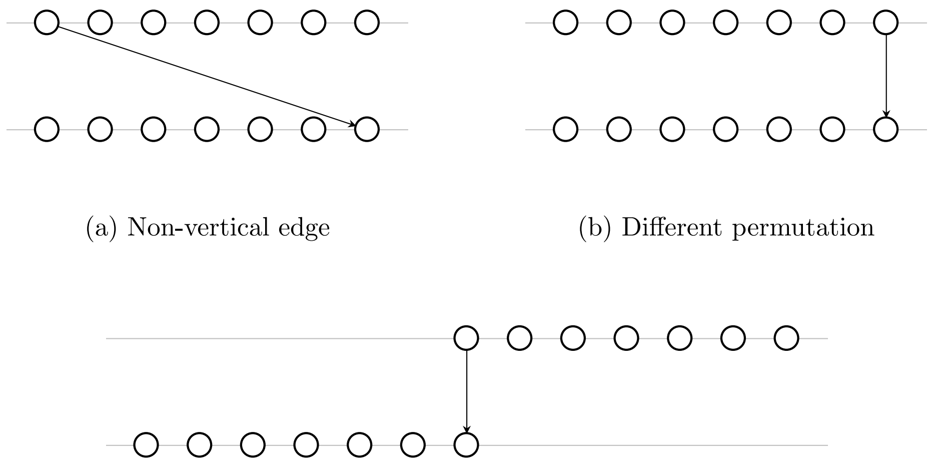
Figure 8.3. Clustered level graph

8.1.6 Non-Connected Graphs

Most graph drawing algorithms assume the input graph for the Sugiyama framework to be connected or do not treat non-connected graphs in a special way. Obviously, a non-connected graph can be drawn by processing its connected components separately and placing the resulting drawings next to each other. However, some problems arise which have to be solved in the hierarchical and cyclic cases.

The first problem is to choose a fitting width for the leveling of each connected component and to determine the starting level for each such component. A leveling algorithm which processes all connected components at the same time would help here. The crossing reduction can be done for each component separately. However, the problem of permuting the components will not be solved by that approach. Different permutations could lead to narrower or wider final drawings. Placing a component between two parts of another component could reduce the size of the used area as well. The final coordinate assignment phase could try to align less edges vertical to get a narrower result for the complete graph.

Survey Figure 8.4 for an example, which is taken from [19]. Figure 8.4(a) shows a possible result of the crossing reduction. Drawing the edge vertically leads to a much higher width of the drawing (see Figure 8.4(c)). Using a non-vertical edge gives the result of Figure 8.4(a). If the crossing reduction were to give the result of Figure 8.4(b), the edge could be drawn vertically with less width. Using a different leveling, e. g., placing each connected component on its own levels would reduce the width as well. Accounting for this general problem would require a careful interweaving of the leveling, the crossing reduction, and the coordinate assignment phases, however.


(a) Non-vertical edge (b) Different permutation

(c) Vertical edge

Figure 8.4. Impact of the phases on the width of a drawing [19]

8.1.7 Width of Dummy Vertices

Another problem regarding the width of a drawing are dummy vertices. Most leveling heuristics ignore dummy vertices when computing the width of a leveling. Others, e. g., [114], consider their width. However, they assume original vertices and dummy vertices to have the same width. But in applications with labeled vertices, the original vertices are much broader than dummy vertices. Most coordinate assignment heuristics do not differentiate between vertices and dummy vertices regarding their width as well. An investigation of such leveling and coordinate assignment algorithms, which allow one size for dummy vertices and a different size for original vertices, is necessary. Obviously, algorithms allowing for a completely arbitrary size of each vertex work here. However, they add unnecessary complexity to the problem.

8.1.8 Incremental Cyclic Level Drawings

An incremental algorithm can handle the addition or removal of vertices or edges by updating the existing result without computing it from scratch. Planarity, e. g., can be tested incrementally [34, 153]. The goal of an incremental drawing algorithm is to retain the mental map. A small modification in the graph itself should yield a small modification in the drawing. The user can recognize the change in the data structure more easily then. There are several incremental drawing algorithms and even incremental versions of the Sugiyama framework [16, 111, 112, 116, 132].

For incremental cyclic level drawings, research is necessary to investigate whether these existing incremental algorithms can be used or completely new ones are needed.

8.1.9 Constraints

One approach for incremental level drawings is to use constraints on the already placed vertices. A constraint can, e. g., imply that one vertex lies on a smaller level than another one, fix the order of two vertices on the same level, or demand that two vertices have the same x-coordinate. Thus, the leveling, crossing reduction, and coordinate assignment phases are affected by constraints. There are several algorithms respecting constraints in the hierarchical Sugiyama framework [56, 132, 150]. In the cyclic Sugiyama framework, new algorithms might be needed to deal with constraints.

8.1.10 Merge of Leveling and Crossing Reduction Phases

The Sugiyama framework splits the problem of finding a nice drawing of a directed graph into four phases. Each of the phases is considered separately. While this reduces the complexity of the problem, it also reduces the quality of the result: In many cases, the third phase could reduce the number of crossings even more if the leveling was not fixed already. Thus, combining the leveling and the crossing reduction phase could lead to better drawings. Utech et al. [149] use an evolutionary algorithm, Chimani et al. [27, 28] propose a planarization approach to tackle this problem. Finding similar results for the cyclic Sugiyama framework remains an open problem.

One possibility for the hierarchical and cyclic framework would be to extend our global sifting approach: Instead of sifting a block only from left to right, we could sift a block, representing an original vertex, in two dimensions and test all levels for the block as well as all positions for the block on each level.

8.1.11 Cyclic Upward Planarity

The algorithms of Chimani et al. [27, 28] mentioned in the previous section, use an upward planarization approach. A graph is upward planar if there exists a planar drawing with all edges pointing upwards. Testing whether a given graph is upward planar is $\mathcal{NP}$-hard [71]. The problem could be extended to cyclic upward planarity. Here, all edges have to point, e. g., counter-clockwise in the 2D drawing. Possible research would be to determine the classes of graphs for which the cyclic upward planarity problem is $\mathcal{NP}$-hard. In contrast to (cyclic) level planarity, in (cyclic) upward planarity the leveling is not yet determined. For radial upward planarity, where all edges have to point away from the center, some research has already been done [36]. Again, combining the concepts of radial and cyclic upward planarity leads to toroidal upward planarity.

8.1.12 Combine Hierarchical and Cyclic Sugiyama framework

Sugiyama [137, 140] suggests for decycling the following method: Compute the strongly connected components of the input graph G and shrink them to one vertex. The resulting graph is acyclic and can be drawn with the hierarchical Sugiyama framework. Obviously, this reduces the complexity of the problem only, if the strongly connected components are rather small. We propose our cyclic variant to draw the non-trivial strongly connected components. See Figure 8.5 for an example.

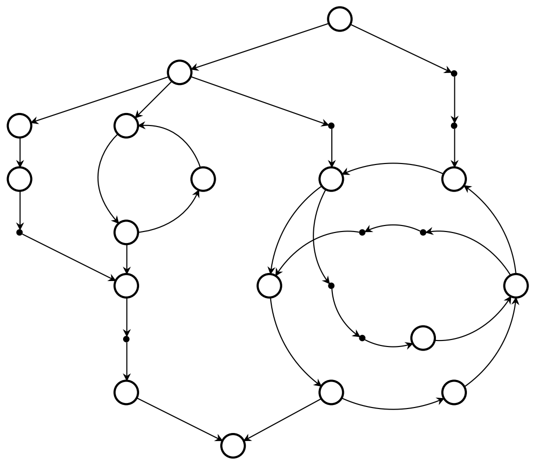


Figure 8.5. Hierarchical drawing with two strongly connected components drawn in a cyclic way.

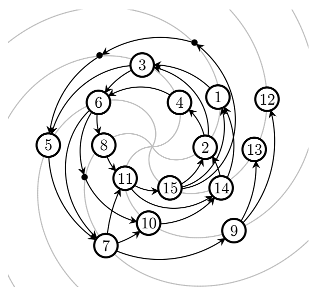
One arising problem is to find suitable positions for vertices in the cyclic drawings which have edges into the hierarchical drawing and vice versa. Techniques from clustered drawings might help here. Combining different drawing algorithms within the same drawing has already been studied [14, 132].

8.1.13 Further 2D Drawings

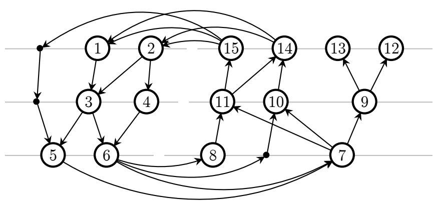
One problem of the cyclic 2D drawing is its area requirement. When traversing the level lines from the center the distance between the lines increases. Hence, much space is wasted in between. Research is needed to find other ways to transform the intermediate drawing to a cyclic 2D drawing. We give two ideas how such transformations could look like.

Figure 8.6(a) presents a cyclic drawing of the example graph G^* using spirals as levels. Hence, the distance between the level lines does not increase when traversing the level lines and much less area is needed. However, the drawing is twisted very much. Some segments are stretched and others shortened. The segment $(7, 11)$ is twisted so much that it now points clockwise. These problems are even worse in larger drawings. However, a drawing in between these two extremes, i. e., using spirals with a smaller curvature as level lines could lead to pleasing drawings within a smaller area.

In Figure 8.6(b) another approach is shown. The intermediate drawing is split into two parts. One is drawn from top to bottom on the left hand side. The second part is rotated by 180° and drawn on the right hand side from bottom to top. Again, the area can be reduced using this approach. However, symmetries in the graph are broken by splitting the drawing.



(a) Levels as spirals



(b) Graph drawn in two directions

Figure 8.6. Transformations of the intermediate drawing of G^* in Figure 1.11(d)

List of Figures

List of Algorithms

Bibliography

[1] N. Alon, C. McDiarmid, and M. Molloy. Edge-disjoint cycles in regular directed graphs. *Journal of Graph Theory*, 22(3):231–237, 1996.

[2] C. Bachmaier. A radial adaption of the sugiyama framework for visualizing hierarchical information. *IEEE Trans. Vis. Comput. 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. Tollis and M. Patrignani, editors, *Proc. Graph Drawing, GD 2008*, volume 5417 of *LNCS*, pages 348–359. Springer, 2009.

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

[7] 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.

[8] 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.

[9] 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.

[10] C. Batini, L. Furlani, and E. Nardelli. What is a good diagram? a pragmatic approach. In *Proc. 4th Internat. Conf. on the Entity Relationship Approach*, pages 312–319. IEEE Computer Society, 1985.

[11] 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.

[12] B. Berger and P. Shor. Approximation algorithms for the maximum acyclic subgraph problem. In *Proc. ACM-SIAM Symposium on Discrete Algorithms, SODA 1990*, pages 236–243, 1990.

[13] J. C. Bermond and C. Thomassen. Cycles in digraphs – a survey. *Journal of Graph Theory*, 5:1–43, 1981.

[14] F. Bertault and M. Miller. An algorithm for drawing compound graphs. In J. Kratochvíl, editor, *Proc. Graph Drawing, GD 1999*, volume 1731 of *LNCS*, pages 197–204. Springer, 1999.

[15] T. C. Biedl, B. P. Madden, and I. G. Tollis. The three-phase method: A unified approach to orthogonal graph drawing. In G. Di Battista, editor, *Proc. Graph Drawing, GD 1997*, volume 1353 of *LNCS*, pages 391–402. Springer, 1997.

[16] K.-F. Böhringer and F. N. Paulisch. Using constraints to achieve stability in automatic graph layout algorithms. In *Proc. ACM Human Factors in Computing Systems Conference, CHI 1990*, pages 43–51, 1990.

[17] K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976.

[18] J. Boyer and W. Myrvold. On the cutting edge: Simplified $\mathcal{O}(n)$ planarity by edge addition. *Journal of Graph Algorithms and Applications*, 8(3):241–273, 2004.

[19] 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, 2001.

[20] J. Branke, S. Leppert, M. Middendorf, and P. Eades. Width-restricted layering of acyclic digraphs with consideration of dummy nodes. *Inform. Process. Lett.*, 81:59–63, 2002.

[21] 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.

[22] C. Buchheim, M. Jünger, and S. Leipert. Improving walker's algorithm to run in linear time. In M. Goodrich, editor, *Proc. Graph Drawing, GD 2002*, volume 2528 of *LNCS*, pages 344–353. Springer, 2002.

[23] L. Carmel, D. Harel, and Y. Koren. Combining hierarchy and energy drawing directed graphs. *IEEE Trans. Vis. Comput. Graphics*, 10(1):46–57, 2004.

[24] M.-J. Carpano. Automatic display of hierarchized graphs for computer-aided decision analysis. *IEEE Trans. Syst., Man, Cybern.*, 10(11):705–715, 1980.

[25] T. Catarci. The assignment heuristic for crossing reduction. *IEEE Trans. Syst., Man, Cybern.*, 25(3):515–521, 1995.

[26] T. M. Chan. A near-linear area bound for drawing binary trees. In *Proc. ACM-SIAM Symposium on Discrete Algorithms, SODA 1999*, pages 161–168, 1999.

[27] 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.

[28] M. Chimani, C. Gutwenger, P. Mutzel, and H.-M. Wong. Upward planarization layout. In D. Eppstein and E. Gansner, editors, *Proc. Graph Drawing, GD 2009*, volume 5849 of *LNCS*, pages 94–106. Springer, 2010.

[29] E. G. Coffman and R. L. Graham. Optimal scheduling for two processor systems. *Acta Informatica*, 1(3):200–213, 1972.

[30] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, 2nd edition, 2001.

[31] H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990.

[32] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.

[33] G. Di Battista and E. Nardelli. Hierarchies and planarity theory. *IEEE Trans. Syst., Man, Cybern.*, 18(6):1035–1046, 1988.

[34] G. Di Battista and R. Tamassia. Incremental planarity testing. In *Proc. Symp. on Foundations of Computer Science, SFCS*, pages 436–441, 1989.

[35] G. Di Battista, R. Tamassia, and I. G. Tollis. Area requirement and symmetry display of planar upward drawings. *Discrete & Computational Geometry*, 7:381–401, 1992.

[36] A. Dolati and S. M. Hashemi. On the sphericity testing of single source digraphs. *Discrete Mathematics*, 308(11):2175–2181, 2008.

[37] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Monographs in Computer Science. Springer, 1999.

[38] S. Dresbach. A new heuristic layout algorithm for directed acyclic graphs. In *Operations Research Proceedings 1994*, pages 121–126, 1995.

[39] T. Dwyer, Y. Koren, and K. Marriott. Drawing directed graphs using quadratic programming. *IEEE Trans. Vis. Comput. Graphics*, 12(4):536–548, 2006.

[40] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

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

[42] P. Eades, Q. Feng, X. Lin, and H. Nagamochi. Straight-line drawing algorithms for hierarchical graphs and clustered graphs. *Algorithmica*, 44:1–32, 2006.

[43] 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.

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

[45] P. Eades, T. Lin, and X. Lin. Two tree drawing conventions. *Internat. J. Comput. Geom. Appl.*, 3(2):133–153, 1993.

[46] P. Eades and X. Lin. A new heuristic for the feedback arc set problem. *Australian Journal of Combinatorics*, 12:15–26, 1995.

[47] P. Eades, X. Lin, and W. F. Smyth. A fast and effective heuristic for the feedback arc set problem. *Inform. Process. Lett.*, 47:319–323, 1993.

[48] P. Eades, X. Lin, and R. Tamassia. An algorithm for drawing hierarchical graphs. *Internat. J. Comput. Geom. Appl.*, 6:145–156, 1996.

[49] P. Eades, B. D. McKay, and N. C. Wormald. On an edge crossing problem. In *Proc. 9th Australian Computer Science Conference*, pages 327–334, 1986.

[50] P. Eades and K. Sugiyama. How to draw a directed graph. *J. Inform. Process.*, 13(4):424–437, 1990.

[51] P. Eades and N. C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11(4):379–403, 1994.

[52] 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.

[53] A. Estrella-Balderrama, J. J. Fowler, and S. G. Kobourov. On the characterization of level planar trees by minimal patterns. In D. Eppstein and E. Gansner, editors, *Proc. Graph Drawing, GD 2009*, volume 5849 of *LNCS*, pages 69–80. Springer, 2010.

[54] S. Even and Y. Shiloach. *NP-completeness of several arrangement problems.* Technical report, Dept. of Computer Science, Technion, Haifa, Israel, 1975.

[55] I. Fáry. On straight-line representations of planar graphs. *Acta Sci. Math. (Szeged)*, 11:229–233, 1948.

[56] I. Finocchi. Layered drawings of graphs with crossing constraints. In J. Wang, editor, *Proc. Computing and Combinatorics, COCOON 2001*, volume 2108 of *LNCS*, pages 357–367. Springer, 2001.

[57] M. M. Flood. Exact and heuristic algorithms for the weighted feedback arc set problem: A special case of the skew-symmetric quadratic assignment problem. *Networks*, 20(1):1–23, 1990.

[58] U. Fößmeier and M. Kaufmann. Drawing high degree graphs with low bend numbers. In F. J. Brandenburg, editor, *Proc. Graph Drawing, GD 1995*, volume 1027 of *LNCS*, pages 254–266. Springer, 1996.

[59] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics.* Addison-Wesley, 2nd edition, 1990.

[60] M. Forster. *Crossings in Clustered Level Graphs.* Dissertation, University of Passau, 2004.

[61] J. J. Fowler and S. G. Kobourov. Minimum level nonplanar patterns for trees. In S.-H. Hong, T. Nishizeki, and W. Quan, editors, *Proc. Graph Drawing, GD 2007*, volume 4875 of *LNCS*, pages 69–75. Springer, 2007.

[62] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.

[63] C. Friedrich and F. Schreiber. Flexible layering in hierarchical drawings with nodes of arbitrary size. In *Proc. of the 27th conference on Australasian computer science*, pages 369–376, 2004.

[64] T. M. J. Fruchterman and R. E. M. Graph-drawing by force directed placement. *Software Pract. Exper.*, 21(11):1129–1164, 1991.

[65] 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.

[66] E. R. Gansner, E. Koutsofios, S. North, and K.-P. Vo. A technique for drawing directed graphs. *IEEE Trans. Software Eng.*, 19(3):214–230, 1993.

[67] C. Gardner, editor. *QFINANCE - The Ultimate Resource.* Berg Publishers, 2009.

[68] M. R. Garey, R. L. Graham, D. S. Johnson, and K. D. E. Complexity results for bandwidth minimization. *SIAM Journal on Applied Mathematics*, 34(3):477–495, 1978.

[69] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freemann, New York, 1979.

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

[71] A. Garg and R. Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM Journal on Computing*, 31(2):601–625, 2001.

[72] M. Gröschel, M. Jünger, and G. Reinelt. On the acyclic subgraph polytope. *Mathematical Programming*, 33(1):28–42, 1985.

[73] A. Gyárfás, J. Komlós, and E. Szemerédi. On the distribution of cycle lengths in graphs. *Journal of Graph Theory*, 8(4):441–462, 1984.

[74] N. G. Hall, T.-E. Lee, and M. E. Posner. The complexity of cyclic shop scheduling problems. *Journal of Scheduling*, 5(4):307–327, 2002.

[75] D. Harel and Y. Koren. Graph drawing by high-dimensional embedding. *Journal of Graph Algorithms and Applications*, 8(2):195–214, 2004.

[76] 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.

[77] P. Healy and A. Kuusik. Algorithms for multi-level graph planarity testing and layout. *Theor. Comput. Sci.*, 320(2–3):331–344, 2004.

[78] P. Healy, A. Kuusik, and S. Leipert. A characterization of level planar graphs. *Discrete Mathematics*, 280:51–63, 2004.

[79] L. S. Heath and S. V. Pemmaraju. Recognizing leveled-planar dags in linear time. In F. J. Brandenburg, editor, *Proc. Graph Drawing, GD 1995*, volume 1027 of *LNCS*, pages 300–311. Springer, 1996.

[80] C. A. R. Hoare. Quicksort. *Computer Journal*, 5(1):10–15, 1962.

[81] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 3rd edition, 2007.

[82] J. E. Hopcroft and R. E. Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, 1974.

[83] F. Hübner. A global approach on crossing minimization in hierarchical and cyclic layouts of leveled graphs. Diploma thesis, University of Passau, 2009.

[84] D. B. Johnson. Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing*, 4(1):77–84, 1975.

[85] 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, *Proc. Graph Drawing, GD 1997*, volume 1353 of *LNCS*, pages 13–24. Springer, 1997.

[86] M. Jünger and S. Leipert. Level planar embedding in linear time. *Journal of Graph Algorithms and Applications*, 6(1):67–113, 2002.

[87] M. Jünger, S. Leipert, and P. Mutzel. Pitfalls of using PQ-trees in automatic graph drawing. In G. Di Battista, editor, *Proc. Graph Drawing, GD 1997*, volume 1353 of *LNCS*, pages 193–204. Springer, 1997.

[88] M. Jünger, S. Leipert, and P. Mutzel. Level planarity testing in linear time. In S. H. Whitesiedes, editor, *Proc. Graph Drawing, GD 1998*, volume 1547 of *LNCS*, pages 224–237. Springer, 1998.

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

[90] M. Jünger and P. Mutzel, editors. *Graph Drawing Software.* Mathematics and Visualization. Springer, 2004.

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

[92] R. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[93] M. Kaufmann and D. Wagner. *Drawing Graphs: Methods and Models*, volume 2025 of *LNCS.* Springer, 2001.

[94] K. Kuratowski. Sur le problème des courbes gauches en topologie. *Fundamenta Mathematicae*, 15:271–283, 1930.

[95] M. Laguna and R. Martí. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11(1):44–52, 1999.

[96] M. Laguna, R. Martí, and V. Valls. Arc crossing minimization in hierarchical digraphs with tabu search. *Computers and Operations Research*, 24(12):1175–1186, 1997.

[97] P. C. B. Lam, W. C. Shui, and W. H. Chan. Characterization of graphs with equal bandwidth and cyclic bandwidth. *Discrete Mathematics*, 242(1–3):283–289, 2002.

[98] S. Lam and R. Sethi. Worst case analysis of two scheduling problems. *SIAM Journal on Computing*, 6:518–536, 1977.

[99] A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. *International Symposium, Rome*, pages 215–232, 1967.

[100] V. Liberatore. Circular arrangements and cyclic broadcast scheduling. *Journal of Algorithms*, 51:185–215, 2004.

[101] Y. Lin. Minimum bandwidth problem for embedding graphs in cycles. *Networks*, 29(3):135–140, 1997.

[102] C. L. Lucchesi. *A Minimax Equality for Directed Graphs.* Dissertation, University of Waterloo, 1976.

[103] E. Mäkinen and M. Sieranta. Genetic algorithms for drawing bipartite graphs. *International Journal of Computer Mathematics*, 53:157–166, 1994.

[104] 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.

[105] L. Medsker and L. C. Jain. *Recurrent Neural Networks: Design and Applications.* The CRC Press International Series on Computational Intelligence. CRC, 1999.

[106] K. Mehlhorn. *Data Structures and Algorithms. Volume 2: Graph Algorithms and $\mathcal{NP}$-Completeness.* EATCS Monographs on Theoretical Computer Science. Springer, 1984.

[107] K. Mehlhorn and W. Rülling. Compaction on the torus. *IEEE Transactions on Computer-Aided Design*, 9(4):389–397, 1990.

[108] S. Meyn and R. L. Tweedie. *Markov Chains and Stochastic Stability.* MIT Press, Cambridge, 2nd edition, 2009.

[109] G. Michal. *Biochemical Pathways (Poster).* Boehringer Mannheim, Penzberg, 1993.

[110] G. Michal, editor. *Biochemical Pathways: An Atlas of Biochemistry and Molecular Biology.* Wiley, 1999.

[111] K. Miriyala, S. W. Hornick, and R. Tamassia. An incremental approach to aesthetic graph layout. In *Proc. Computer-Aided Software Engineering, CASE 1993*, pages 297–308, 1993.

[112] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *Visual Languages and Computing*, 6:183–210, 1995.

[113] 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.

[114] N. S. Nikolov, A. Tarassov, and J. Branke. In search for efficient heuristics for minimum-width graph layering with consideration of dummy nodes. *ACM Journal of Experimental Algorithmics*, 10(2.7):1–27, 2005.

[115] J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer Series in Operations Research and Financial Engineering. Springer, 2nd edition, 2006.

[116] S. C. North. Incremental layout in DynaDAG. In F. J. Brandenburg, editor, *Proc. Graph Drawing, GD 1995*, volume 1027 of *LNCS*, pages 409–418. Springer, 1996.

[117] F. N. Paulish. *The Design of an Extendible Graph Editor*, volume 704 of *LNCS*. Springer, 1993.

[118] J. L. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice-Hall, 1981.

[119] C. Pich. Drawing directed graphs clockwise. In D. Eppstein and E. Gansner, editors, *Proc. Graph Drawing, GD 2009*, volume 5849 of *LNCS*, pages 369–380. Springer, 2010.

[120] R. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389–1401, 1957.

[121] 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.

[122] H. C. Purchase, D. Carrington, and J.-A. Allder. Empirical evaluation of aesthetics-based graph layout. *Empirical Software Engineering*, 7(3):233–255, 2002.

[123] H. C. Purchase, R. F. Cohen, and J. M. Validating graph drawing aesthetics. In F. J. Brandenburg, editor, *Proc. Graph Drawing, GD 1995*, volume 1027 of *LNCS*, pages 435–446. Springer, 1996.

[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 Trans. Syst., Man, Cybern.*, 18(1):2–8, 1988.

[126] E. M. Reingold and J. S. Tilford. Tidier drawings of trees. *IEEE Trans. Software Eng.*, 7(2):223–228, 1981.

[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 Computer Society Press, 1993.

[128] 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.

[129] 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.

[130] G. Sander. Graph layout for applications in compiler construction. *Theor. Comput. Sci.*, 217(2):175–214, 1999.

[131] W. Schnyder. Embedding planar graphs on the grid. In *Proc. ACM-SIAM Symposium on Discrete Algorithms, SODA 1990*, pages 138–148, 1990.

[132] F. Schreiber. *Visualisierung biochemischer Reaktionsnetze.* Dissertation, University of Passau, 2001.

[133] A. Schrijver. *Theory of Linear and Integer Programming.* Wiley-Interscience, 1986.

[134] P. Serafini and W. Ukovich. A mathematical model for periodic scheduling problems. *SIAM Journal of Discrete Mathematics*, 2(4):550–581, 1989.

[135] W.-K. Shih and W.-L. Hsu. A new planarity test. *Theor. Comput. Sci.*, 223(1–2):179–191, 1999.

[136] H. Stamm. On feedback problems in planar digraphs. In *Graph-Theoretic Concepts in Computer Science*, volume 484 of *LNCS*, pages 79–89. Springer, 1991.

[137] K. Sugiyama. *Graph Drawing and Applications for Software and Knowledge Engineers*, volume 11 of *Software Engineering and Knowledge.* World Scientific, 2002.

[138] K. Sugiyama and K. Misue. Visualization of structural information: Automatic drawing of compound digraphs. *IEEE Trans. Syst., Man, Cybern.*, 21(4):876–892, 1991.

[139] K. Sugiyama and K. Misue. Graph drawing by the magnetic spring model. *Journal of Visual Languages and Computing*, 6:217–231, 1995.

[140] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Syst., Man, Cybern.*, 11(2):109–125, 1981.

[141] R. Tamassia and I. G. Tollis. A unified approach to visibility representations of planar graphs. *Discrete & Computational Geometry*, 1:321–341, 1986.

[142] R. Tamassia and I. G. Tollis. Planar grid embedding in linear time. *IEEE Trans. Circuits and Systems*, 36(9):1230–1234, 1989.

[143] R. Tamassia and I. G. Tollis. Representations of graphs on a cylinder. *SIAM Journal of Discrete Mathematics*, 4(1):139–149, 1991.

[144] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[145] C. Thomassen. Disjoint cycles in digraphs. *Combinatorica*, 3(3–4):393–396, 1983.

[146] V. Tsiaras, S. Triantafilou, and I. G. Tollis. DAGmaps: Space filling visualization of directed acyclic graphs. *Journal of Graph Algorithms and Applications*, 13(3):319–347, 2009.

[147] E. R. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, 2nd edition, 2001.

[148] J. D. Ullman. NP-complete scheduling problems. *Journal of Computer and System Sciences*, 10:384–393, 1975.

[149] J. Utech, J. Branke, H. Schmeck, and P. Eades. An evolutionary algorithm for drawing directed graphs. In *Proc. Imaging Science, Systems, and Technology*, pages 154–160, 1998.

[150] V. Waddle. Graph layout for displaying data structures. In J. Marks, editor, *Proc. Graph Drawing, GD 2000*, volume 1984 of *LNCS*, pages 98–103. Springer, 2001.

[151] C. Ware, H. C. Purchase, L. Colpoys, and M. McGill. Cognitive measurements of graph aesthetics. *Information Visualization*, 1(2):103–110, 2002.

[152] J. Warfield. Crossing theory and hierarchy mapping. *IEEE Trans. Syst., Man, Cybern.*, 7:502–523, 1977.

[153] J. Westbrook. Fast incremental planarity testing. In *Proc. of the 19th International Colloquium on Automata, Languages and Programming*, volume 623 of *LNCS*, pages 342–353. Springer, 1992.

Index