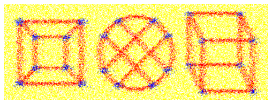


Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 15, no. 5, pp. 631–659 (2011)

Global k-Level Crossing Reduction

Christian Bachmaier[1] *Franz J. Brandenburg*[1] *Wolfgang Brunner*[1]
Ferdinand Hübner[2]

[1]University of Passau, 94030 Passau, Germany
[2]Bedag Informatik AG, 3011 Bern, Switzerland

Abstract

Directed graphs are commonly drawn by a four phase framework introduced by Sugiyama et al. in 1981. The vertices are placed on parallel horizontal levels. The edge routing between consecutive levels is computed by solving one-sided 2-level crossing minimization problems, which are repeated in up and down sweeps over all levels. Crossing minimization problems are generally $\mathcal{NP}$-hard.

We introduce a global crossing reduction, which at any particular time considers all crossings between all levels. Our approach is based on the sifting technique. It yields an improvement of 5 – 10% in the number of crossings over the level-by-level one-sided 2-level crossing reduction heuristics. In addition, it avoids type 2 conflicts which are crossings between edges whose endpoints are dummy vertices. This helps straightening long edges spanning many levels. Finally, the global crossing reduction approach can directly be extended to cyclic, radial, and clustered level graphs achieving similar improvements. The running time is quadratic in the size of the input graph, whereas the common level-by-level approaches are faster but operate on larger graphs with many dummy vertices for long edges.

Submitted:	Reviewed:	Revised:	Accepted:
April 2010	October 2010	November 2010	April 2011
	Final:	Published:	
	Mai 2011	October 2011	
Article type:		Communicated by:	
Regular paper		M. S. Rahman and S. Fujita	

Supported by the Deutsche Forschungsgemeinschaft (DFG), grant Br835/15-1.

A preliminary version [3] was presented at the Workshop on Algorithms and Computation, WALCOM 2010.

E-mail addresses: bachmaier@fim.uni-passau.de (Christian Bachmaier) brandenb@informatik.uni-passau.de (Franz J. Brandenburg) brunner@fim.uni-passau.de (Wolfgang Brunner) ferdinand.huebner@bedag.ch (Ferdinand Hübner)

1 Introduction

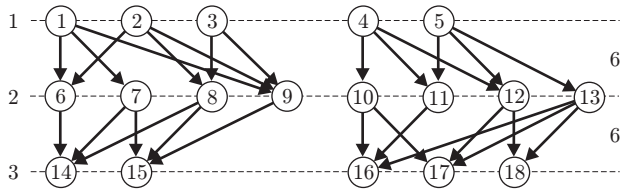
The four phase framework introduced by Sugiyama et al. [43] is the standard algorithm for drawing directed graphs. It displays them in a hierarchical manner and operates in four phases: cycle removal, leveling, crossing reduction, and coordinate assignment. First, it reverses appropriate edges to eliminate cycles. Then, it assigns vertices to levels, which are their y-coordinates, and introduces dummy vertices splitting long edges at their crossings with spanned levels. This results in a proper k-level graph. In the third phase the vertices are permuted within the levels to reduce edge crossings. Finally, the x-coordinates are computed such that all vertices have integral coordinates and the drawing meets some aesthetic criteria such as few bends per edge. Typical applications for such drawings are schedules, UML diagrams, and flow charts, where temporal or causal dependencies are modeled by directed edges and are expressed by a left to right or a downward direction, see [14, 31, 43].

In this paper we focus on the crossing reduction phase, where the vertices on each level are rearranged to minimize the total number of crossings. The common solution to this $\mathcal{NP}$-hard *k-level crossing minimization problem* [25] is a reduction to the *one-sided 2-level crossing minimization problem*, which is solved repeatedly in several up and down sweeps [31, 43]. In a down sweep the vertices V_{i-1} in the upper level are fixed and the vertices V_i in the lower level are reordered reducing the local number of edge crossings between the two levels. In an up sweep the roles of the levels are switched. The one-sided 2-level crossing minimization problem is $\mathcal{NP}$-hard [19], even for forests of 4-stars [36]. However, there are many heuristics for this problem [31]. Small instances can be solved exactly by an ILP approach [30]. There are no reasonable approximations for the k-level crossing minimization problem. The ratios from the one-sided 2-level crossing minimization problem [19, 37] do not translate to the general problem.

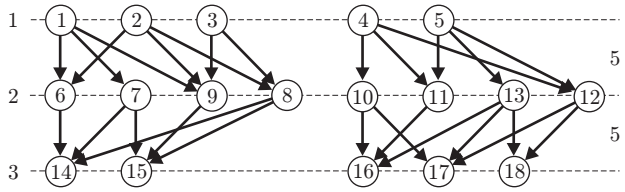
An important feature of crossing reduction algorithms is the absence of *type 2 conflicts*, which are crossings of two edges between dummy vertices. Among others, our favored fourth phase algorithm of Brandes and Köpf [10] assumes the absence of type 2 conflicts. It aligns long edges vertically and so meets an important aesthetic criterion [31] for nice hierarchical drawings with at most two bends per edge.

The *barycenter* and the *median heuristic* [31] are two common 2-level crossing reductions. They place each vertex $v \in V_i$ at the barycenter or median position of its predecessors in V_{i-1}. Then, V_i is sorted by these values. So the edges shall be short and induce few crossings. These techniques are simple, fast, and avoid type 2 conflicts, but they often leave too many crossings.

In 2-level algorithms the number of crossings between levels V_i and V_{i+1} and thus the total number of crossings can increase while permuting V_i for the 2-level crossing reduction between V_{i-1} and V_i. So, such heuristics push the crossings downwards or upwards until they are resolved at the extreme levels. An immediate extension is the *centered 3-level crossing reduction*. Here, V_i is permuted while keeping the orders of V_{i-1} and V_{i+1} fixed and considering the crossings above and below level i. However, this introduces type 2 conflicts.



(a) Optimal level-by-level order with 12 crossings



(b) Optimal order with 10 crossings

Figure 1: Crossing reduction using an exact level-by-level sweep method has been stuck in a local minimum

All these approaches suffer from a general problem: They have a local view to the crossing minimization problem. So they tend to get stuck in a local optimum. Bastert and Matuszewski claim [31] that the results of a 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" [31, page 102]. After empirical tests of several one-sided 2-level approaches Stallmann et al. [42, page 32] drew the conclusion that "the most pressing issue from a practical perspective is a generalization to $k > 2$ levels". Our approach addresses this gap. Fig. 1 shows an example, where even an optimal one-sided 2-level crossing reduction gets stuck in a local minimum. In the left component of Fig. 1(a) an optimal top-down sweep swaps vertices 8 and 9, which reduces the number of crossings by two. However, the subsequent bottom-up sweep will undo this change. Since the right component is symmetric, the graph has 12 crossings independently of the direction of the final sweep. Our algorithm solves this example with the optimal solution with 10 crossings as shown in Fig. 1(b).

Tutte's algorithm [18] can be seen as a first approach with a global view, but it does not address crossings directly. Its quality concerning the number of crossings is open, as Eades and Sugiyama [18] state in Problem 8. Here, the positions of the vertices on the extreme levels are fixed in any order. For the vertices in the other levels the x-coordinate is chosen as the weighted average of the x-positions of all its neighbors. This is modeled by a sparse system of linear equations.

Sifting is a modification of sorting by insertion and it has been used successfully for vertex minimization in ordered binary decision diagrams [40]. Later it has been adapted to the one-sided 2-level crossing reduction problem [34]. The idea is to keep track of the number of crossings while in a *sifting step* a vertex $v \in V_i$ is moved along a fixed order of the vertices in V_i. Finally, v is placed at its locally best position. The method is an extension of the greedy-switch heuristic [17], where v is swapped iteratively with its current successor. We call a single swap a *sifting swap* and the execution of a sifting step for every vertex in V_i a *sifting round*. In general, sifting leaves fewer crossings than the simple one-sided 2-level heuristics at the expense of a higher running time and potential type 2 conflicts [31].

Matuszewski et al. [34] have extended sifting towards a more global view, which we call *ordered k-level sifting* to avoid confusion. There, the vertices are sorted by their degree and are first sifted in increasing order and then in decreasing order. For swapping two consecutive vertices v and w their incident edges to and from neighbors on both adjacent levels are taken into account. The heuristic does not sweep level-by-level but is still limited to a local view, since long edges are not treated as a whole. Our *centered 3-level sifting* does the same with the vertices ordered level-by-level. Both algorithms yield similar results. Jünger et al. [29] have developed an exact ILP approach for the $\mathcal{NP}$-hard k-level crossing minimization, which can be used in practice for small graphs. Moreover, metaheuristics have been proposed in the literature, such as genetic algorithms [32,45], tabu search [33], or windows optimization [22]. These general (stochastic) global search approaches usually compute good solutions with few crossings at the expense of high running times.

In this paper we propose a new and global crossing reduction technique. The algorithm yields better results than the common heuristics. It is directly extendable to more general crossing reduction problems, avoids type 2 conflicts, and runs in quadratic time in the size of the graph. This distinguishes our approach from most 2-level approaches which extensively use dummy vertices, whose number is up to $\mathcal{O}(k \cdot |E|) \subseteq \mathcal{O}(|V|^3)$. Note that the edge bundling technique of Eiglsperger et al. [21] groups dummy vertices horizontally in each level. This improves the running time of the sweeping barycenter and median heuristics to be independent of the number of dummy vertices. However, their approach cannot be used for more advanced heuristics like sifting.

Recently, Chimani et al. [11, 12] presented an advanced approach based on upward planarization which combines the leveling and the crossing reduction phases of the hierarchical framework. Generally, from an algorithm and software engineering perspective the subdivision of a complicated problem into disjoint phases as done in the hierarchical framework makes sense. However, for obtaining global optima the phases cannot be treated independently, as is done traditionally and also in this paper. For example, the number of crossings clearly depends on the leveling. For a competitive extension of global sifting which performs leveling and crossing reduction simultaneously see [7].

The remainder of the paper is organized as follows: After introducing some notation and previous results in the next section, we present our new crossing

reduction approach and analyze its complexity in Sect. 3. Then, in Sect. 4, we show benchmarks which empirically compare our algorithm with established strategies. Afterwards, we present some applications of the block concept on related crossing optimization and reduction methods in Sect. 5. Finally, we summarize the results and discuss some open problems in Sect. 6.

2 Preliminaries

We suppose that a directed graph without self-loops has passed the cycle removal and leveling phases of the four phase framework. The outcome is a *k-level graph* $G = (V, E, \phi)$, where $\phi\colon V \to \{1, 2, \ldots, k\}$ is a surjective level assignment of the vertices with $\phi(u) < \phi(v)$ for each edge $(u, v) \in E$. For an edge $e = (u, v) \in E$ we define its length as $\text{span}(e) := \phi(v) - \phi(u)$. An edge e is *short* if $\text{span}(e) = 1$ and *long* otherwise. A graph is *proper* if all edges are short. Each 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. Let $G' = (V', E', \phi')$ denote the proper version of G. As in [10] short edges are called *segments* of e. The first and the last segments of an edge are the *outer segments* and the others the *inner segments*. Inner segments connect two dummy vertices.

Consider the proper level graph G'. For a vertex v we denote the set of neighbors from incoming and outgoing segments by $N^-(v) := \{\, u \in V' \mid (u, v) \in E' \,\}$ and $N^+(v) := \{\, w \in V' \mid (v, w) \in E' \,\}$, respectively. G' is *ordered* if the vertices in 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 order for each level. This induces an order of the sets $N^-(\cdot)$ and $N^+(\cdot)$. Let $\deg^-(v) := |N^-(v)|$ and $\deg^+(v) := |N^+(v)|$ denote the indegree and outdegree of a vertex v and set $\deg(v) := \deg^-(v) + \deg^+(v)$. In an ordered level graph two segments are in *conflict* if they cross or share a vertex. Conflicts are of *type 0*, *1*, or *2* if they are induced by 0, 1, or 2 inner segments, respectively.

Next, we define blocks, which prevent dealing with dummy vertices. Thus, the running time of our algorithm is measured in the size of the input graph and is independent of the number of dummy vertices. A *block* is a single vertex of V or a maximum connected subgraph of dummy vertices, i. e., the inner segments of a long edge. The blocks represent the vertices of a graph related to G', where the edges are the outer segments. We will always denote (dummy) vertices by lower case letters and blocks by upper case letters. For a block A define $x = \text{upper}(A)$ ($y = \text{lower}(A)$) to be the unique vertex x in A (y in A) with no incoming (outgoing) segments in A. x and y always exist but may coincide.

Let $N^-(A) := N^-(\text{upper}(A))$, $N^+(A) := N^+(\text{lower}(A))$, $\deg(A) := |N^-(A)| + |N^+(A)|$, and let $\text{levels}(A)$ be the set of level numbers which contain (dummy) vertices of A. With $\text{block}(v)$ we denote the block of the vertex $v \in V'$. Let $\mathcal{B}$ be an arbitrarily 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 order. We use the binary relation $\prec$ with $A \prec B \Leftrightarrow \pi(A) < \pi(B)$ for comparing blocks $A, B \in \mathcal{B}$ according to their positions π.

2.1 One-Sided 2-Level Sifting

In a sifting step the locally best position for a vertex is sought. Therefore, all positions in its level are tested sequentially, where the cost criterion is the number of crossings. For its computation we adapt the idea of Baur and Brandes [8], which was introduced for *circular crossing reduction*.

Consider a sifting step for a 2-level graph $G = (V_1 \dot\cup V_2, E, \phi)$ with the vertex orders $\pi_i \colon V_i \to \{1, \dots, |V_i|\}$ in levels $i \in \{1, 2\}$ with π_1 fixed. To determine a locally optimal position of a vertex $v \in V_2$ it is sufficient to record the change in the number of crossings while swapping v with its current consecutive successor $w \in V_2$. For this, only edges incident to v or w must be considered. After a swap exactly those pairs of these edges cross which did not cross before. All other crossings remain unchanged. Let $\chi(\pi_2)$ be the number of crossings of an order (π_1, π_2) of G and $\chi_{\pi_2}(e, f) \in \{0, 1\}$ be the number of crossings between two edges $e, f \in E$. Then, we obtain Lemma 1 by a direct adaption from [8], which formalizes the change in the number of crossings per swap. At the end of one step v is placed where the intermediary crossing counts reached their minimum.

Lemma 1 (Bauer, Brandes) *Let $v, w \in V_2$ with $\pi_2(v) = \pi_2(w) - 1$ be consecutive vertices of a 2-level graph $G = (V_1 \dot\cup V_2, E, \phi)$ in an order (π_1, π_2) and let (π_1, π_2') be the order with their positions swapped, then*

$$\chi(\pi_2') = \chi(\pi_2) - c_{\pi_2}(v, w) + c_{\pi_2'}(v, w),$$
$$\text{where} \quad c_{\pi_2}(v, w) = \sum_{u \in N^-(v)} \sum_{x \in N^-(w)} \chi_{\pi_2}\left((u, v), (x, w)\right) \quad .$$

3 Global Sifting

A major drawback of the established crossing reduction algorithms is their local view. We present a new approach using ideas from [10] and [21], which avoids type 2 conflicts. Eiglsperger et al. [21] have used 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. This improves only the running time but not the quality of the result. We treat all dummy vertices of an edge and each original vertex as a block and find the locally best position for the whole block in one sifting step. This eliminates a problem of 2-level approaches which lack this global view on crossings of long edges. In the initialization the list of blocks $\mathcal{B}$ is sorted arbitrarily and each block A receives its position $\pi(A)$ in $\mathcal{B}$ (line 1 in Algorithm 1). At any time during the execution of the algorithm we interpret $\pi(A)$ as an x-coordinate of each vertex v in the block A and $\phi(v)$ as its y-coordinate. This results in a drawing respecting the current order of $\mathcal{B}$. All vertices of a block get the same x-coordinate. Hence, the order is type 2 conflict free. These are important properties of Algorithm 1.

The main part of the algorithm is the sifting step (line 4). There, all positions for a block A are tested and A is moved to the position with the fewest crossings.

This is done for each block $A \in \mathcal{B}$ (line 3) and repeated a certain number of times (line 2). Alternatively, one may repeat until the improvement is below a given threshold. Our experiments have shown that ten rounds suffice to reach such a situation. Finally, each vertex is set to the position of its block (line 5), the vertices in each level are sorted according to these positions (line 6). Finally, the graph is returned (line 7).

Algorithm 1: GLOBAL-SIFTING

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

```
1 create list B of all blocks in G'
2 for 1 ≤ i ≤ ρ do
3 |   foreach A ∈ B do
4 |   |   B ← SIFTING-STEP(G', B, A)
5 foreach v ∈ V' do π(v) ← π(block(v))
6 sort vertices in each level according to π
7 return G'
```

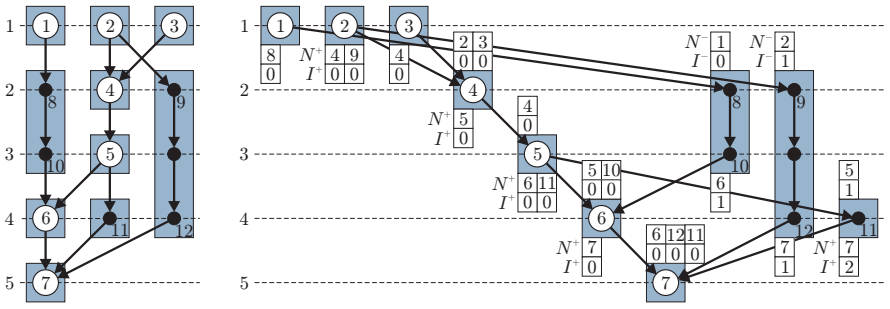
3.1 Building the Block List

The graph is partitioned into blocks. Each block A gets an arbitrary but unique position $\pi(A)$ in the block list $\mathcal{B}$. As an example consider Fig. 2(a). The input graph with 7 vertices has 6 dummy vertices, which are drawn as black circles. The dummy vertices are combined into 3 blocks and each original vertex forms its own block. The 10 resulting blocks are shown in Fig. 2(b) with an arbitrary order π.

If a given order (without type 2 conflicts) shall only be improved by global sifting in a postprocessing step, a straightforward initialization strategy is to topologically sort the blocks according to the order in the levels from left to right in $\mathcal{O}(|E'|)$. Our experiments have shown that a good initial order of the blocks leads to better results. However, these can also be achieved by one or two additional sifting rounds.

3.2 Initialization of a Sifting Step

To improve the performance of one sifting step [8] it is necessary to keep the adjacency lists $N^-(A)$ and $N^+(A)$ of each block $A \in \mathcal{B}$ sorted according to ascending positions of the neighboring blocks in $\mathcal{B}$. We store them in arrays for random access. Additionally, we store two index arrays $I^-(A) = I^-(\text{upper}(A))$ and $I^+(A) = I^+(\text{lower}(A))$ of lengths $|I^-(A)| := |N^-(A)|$ and $|I^+(A)| := |N^+(A)|$, respectively. As an inverted file $I^-(A)$ stores the indices where upper(A) is stored in each adjacent block B's adjacency array $N^+(B)$. More precisely, let $b = N^-(A)[i]$ be a neighbor of upper(A) with $B = \text{block}(b)$. Then, $I^-(A)[i]$



(a) A level graph with ten blocks

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

Figure 2: Blocks as sifting objects

holds the index where upper(A) is stored in $N^+(B) = N^+(b)$. Symmetrically, $I^+(A)$ stores the indices where lower(A) is stored in the adjacency $N^-(B)$ of each adjacent block B. See Fig. 2(b) for an example and consider, e. g., block(11) in detail: $N^+(\text{block}(11))[0] = 7$ indicates that vertex 7 is the first neighbor (stored at position 0) of 11 connected by an outgoing edge. The corresponding value $I^+(\text{block}(11))[0] = 2$ indicates, that vertex 11 itself is stored at position 2 in the incoming adjacency list of its neighbor block(7), i. e., $N^-(\text{block}(7))[2] = 11$. $N^-(\text{block}(11))[0] = 5$ stores the incoming adjacency 5 at its first (and only) position 0. The corresponding value $I^-(\text{block}(11))[0] = 1$ indicates that 11 itself is stored at position 1 in the adjacency of its neighbor 5, i. e., $N^+(\text{block}(5))[1] = 11$. For each block the creation of the four arrays (line 2 of Algorithm 3) can be done in $\mathcal{O}(|E|)$ time as Algorithm 2 shows: Traverse the blocks A in the current order of $\mathcal{B}$ and insert upper(A) (lower(A)) at the next free position j of the cleared adjacency array $N^+(\text{lower}(B))$ ($N^-(\text{upper}(B))$) of each incoming (outgoing) neighbor B. Both values for $I^+(B)$ and $I^-(A)$ ($I^-(B)$ and $I^+(A)$) and their positions are only known after the second traversal of a segment s. Thus, we cache the first array position j as an attribute p of s. Benchmarks [28] have shown that there is a considerable speed-up if only those adjacencies are updated that are disarranged after a sifting step instead of completely sorting all adjacency lists. The theoretical running time is not affected by this improvement, however.

3.3 Sifting Step

In a sifting step by Algorithm 3 all positions p in $\mathcal{B}$ are tested for a block $A \in \mathcal{B}$ (lines 5–8) and, then, A is moved to the position p^* which has caused the least number of crossings. Note that it is not necessary to count the crossings for each position of A. As in [8] and in contrast to other sifting algorithms, which always maintain the absolute number of crossings, we treat the number of crossings of

Algorithm 2: SORT-ADJACENCIES

Input: Proper k-level graph $G' = (V', E', \phi')$, ordered list $\mathcal{B}$ of blocks in G'

Output: Ordered sets $N^{\cdot}(A)$ and $I^{\cdot}(A)$ for each block $A \in \mathcal{B}$

```
1  for i ← 0 to |B| − 1 do
2      π(B[i]) ← i
3      clear the arrays N⁻(B[i]), N⁺(B[i]), I⁻(B[i]), and I⁺(B[i])
4  foreach A ∈ B do
5      foreach s ∈ { (u, v) ∈ E' | v = upper(A) } do
6          insert v at the next free position j of N⁺(u)
7          if π(A) < π(block(u)) then p[s] ← j      // first traversal of s
8          else I⁺(u)[j] ← p[s]; I⁻(v)[p[s]] ← j    // second traversal of s
9      foreach s ∈ { (w, x) ∈ E' | w = lower(A) } do
10         insert w at the next free position j of N⁻(x)
11         if π(A) < π(block(x)) then p[s] ← j      // first traversal of s
12         else I⁻(x)[j] ← p[s]; I⁺(w)[p[s]] ← j    // second traversal of s
```

A as $\chi = 0$ when A is placed to the first position. Thereafter, we only compute the change in the number of crossings when A is iteratively swapped with its right neighbor (line 6).

Algorithm 3: SIFTING-STEP

Input: Proper k-level graph $G' = (V', E', \phi')$, ordered list $\mathcal{B}$ of blocks in G', block $A \in \mathcal{B}$ to sift

Output: Updated order of $\mathcal{B}$

```
1  B' ← A ≺ B[0] ≺ ··· ≺ B[|B| − 1]    // new order B' with A put to front
2  SORT-ADJACENCIES(G', B')
3  χ ← 0; χ* ← 0                       // current and best number of crossings
4  p* ← 0                              // best block position
5  for p ← 1 to |B'| − 1 do
6      χ ← χ + SIFTING-SWAP(A, B'[p])
7      if χ < χ* then
8          χ* ← χ; p* ← p
9  return B'[1] ≺ ··· ≺ B'[p*] ≺ A ≺ B'[p* + 1] ≺ ··· ≺ B'[|B'| − 1]
```

3.4 Sifting Swap

The sifting swap computes the change in the number of crossings when a block A is swapped with its right successor B in $\mathcal{B}$. In contrast to one-sided crossing reduction, our global approach takes the whole neighborhood of both blocks

into account when it computes the change in the number of crossings. Lemma 2 states which segments are involved.

Lemma 2 *Let $\mathcal{B}$ be the block list in the current order. Let $B \in \mathcal{B}$ be the successor of $A \in \mathcal{B}$. If swapping A and B changes the crossings between two segments, then one of them is an incident outer segment of A or B and the other segment is an incident outer segment of the same kind (incoming or outgoing) or an inner segment of the other block.*

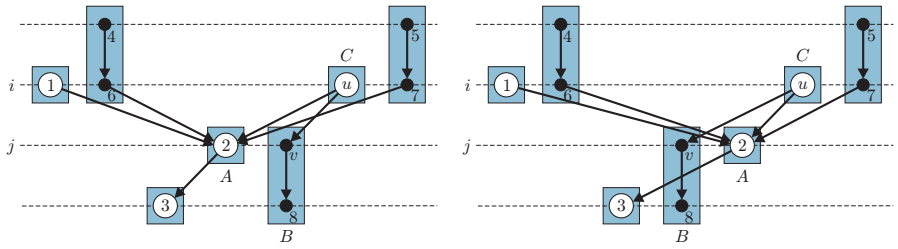
Proof: Note that only segments between the same levels can cross. As no type 2 conflicts occur, at least one of the segments of a crossing has to be an outer segment. Let (a, b) and (c, d) be two segments between the same levels with $a \neq c$ and $b \neq d$. If the two segments cross after swapping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

The following fact is easily seen and the resulting changes in the number of crossings are obvious.

Fact 1 *Let $\mathcal{B}$ be the block list in the current order. Let $B \in \mathcal{B}$ be the successor of $A \in \mathcal{B}$. Let i and j be the two levels framing the incoming outer segments of A (the other three cases are symmetric). If there is a segment (u, v) between i and j which is either an incoming outer segment of B or an inner segment of B, then the incoming segments of A starting at a block left of* block(u) *cross (u, v) only after the swap of A and B, the segments starting at* block(u) *never cross (u, v), and the segments starting right of* block(u) *cross (u, v) only before the swap. There are no other changes among the crossings due to Lemma 2.*

For an illustration consider Fig. 3. The incoming segment (u, v) of block B starts at block C. Thus, all incoming segments of A starting at a block left of C, namely $(1, 2)$ and $(6, 2)$, cross (u, v) only after the swap of A and B. The segment connecting blocks C and A, i. e., $(u, 2)$, never crosses (u, v) and the incoming segments of A starting right of block C, namely $(7, 2)$, cross (u, v) only before the swap. The outgoing segment $(2, 3)$ of A crosses the inner segment $(v, 8)$ of B only after the swap.

Algorithm 4 shows the details of a sifting swap. First, the levels at which (significant) swaps occur and the direction of the segments changing their crossings are found (lines 2–6). For each entry (l, d) of the set $\mathcal{L}$ the two vertices a and b of A and B on level l are retrieved.



(a) Current block order

(b) Blocks A and B swapped

Figure 3: Changes in crossings for a swap

Algorithm 4: SIFTING-SWAP

Input: Consecutive blocks A, B in the ordered blocklist $\mathcal{B}$
Output: Change in crossing count

```
1  begin
2      L ← ∅; Δ ← 0                    // L is a set and, thus, duplicate free
3      if φ(upper(A)) ∈ levels(B) then L ← L ∪ {(φ(upper(A)), −)}
4      if φ(lower(A)) ∈ levels(B) then L ← L ∪ {(φ(lower(A)), +)}
5      if φ(upper(B)) ∈ levels(A) then L ← L ∪ {(φ(upper(B)), −)}
6      if φ(lower(B)) ∈ levels(A) then L ← L ∪ {(φ(lower(B)), +)}
7      foreach (l, d) ∈ L do
8          let a in A and b in B be the vertices with φ(a) = φ(b) = l
9          Δ ← Δ + USWAP(a, b, N^d(a), N^d(b))
10         UPDATE-ADJACENCY(a, b, N^d(a), I^d(a), N^d(b), I^d(b))
11     swap positions of A and B in B; π(A) ← π(A) + 1; π(B) ← π(B) − 1
12     return Δ
```

When swapping A and B only a and b are swapped in their level and no order changes in the level of their neighbors $N^d(a)$ and $N^d(b)$. Thus, the computation of the change in the number of crossings can be done as in [8] and is described in Algorithm 5: The neighbors are traversed from left to right. If a neighbor of a is found (lines 5 and 6), its segment will cross all remaining $s-j$ incident segments of b after the swap. If a neighbor of b is found (lines 7 and 8), its segment has crossed all remaining $r-i$ incident segments of a before the swap. Common neighbors present both cases at the same time (line 9).

Algorithm 5: USWAP

Input: Consecutive vertices $a, b \in V$, $N^d(a), N^d(b) \in V$
Output: Change in crossing count

```
1  let x_0 ≺ ··· ≺ x_{r-1} ∈ N^d(a) be the neighbors of a in direction d
2  let y_0 ≺ ··· ≺ y_{s-1} ∈ N^d(b) be the neighbors of b in direction d
3  c ← 0; i ← 0; j ← 0
4  while i < r and j < s do
5  |  if π(block(x_i)) < π(block(y_j)) then
6  |  |  c ← c + (s − j); i ← i + 1
7  |  else if π(block(x_i)) > π(block(y_j)) then
8  |  |  c ← c − (r − i); j ← j + 1
9  |  else c ← c + (s − j) − (r − i); i ← i + 1; j ← j + 1
10 return c
```

An update of the adjacency after a swap (line 10) is necessary if a and b have common neighbors. Algorithm 6 shows how this can be done in overall $\mathcal{O}(\deg(A) + \deg(B))$ time similarly to the crossing counting function Algorithm 5.

3.5 Time Complexity

Lemma 3 *Let $G = (V, E, \phi)$ be a level graph. Then, $\sum_{B \in \mathcal{B}} \deg(B) \leq 4 \cdot |E|$.*

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

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

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

Algorithm 6: UPDATE-ADJACENCIES

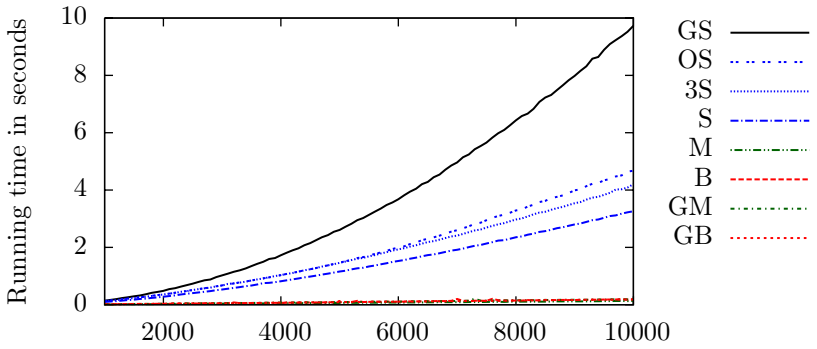
Input: Vertices $a, b \in V'$, $N^d(a), I^d(a), N^d(b), I^d(b)$
Output: Updated adjacencies of a and b and all common neighbors

```
1  let x_0 ≺ ··· ≺ x_{r-1} ∈ N^d(a) be the neighbors of a in direction d
2  let y_0 ≺ ··· ≺ y_{s-1} ∈ N^d(b) be the neighbors of b in direction d
3  i ← 0; j ← 0
4  while i < r and j < s do
5      if π(block(x_i)) < π(block(y_j)) then i ← i + 1
6      else if π(block(x_i)) > π(block(y_j)) then j ← j + 1
7      else
8          z ← x_i                    // = y_j
9          swap entries at pos. I^d(a)[i] and I^d(b)[j] in N^{-d}(z) and in I^{-d}(z)
10         I^d(a)[i] ← I^d(a)[i] + 1; I^d(b)[j] ← I^d(b)[j] − 1
11         i ← i + 1; j ← j + 1
```

4 Experimental Results

For the sake of completeness we have extended the barycenter and median crossing reduction strategies to blocks as well. We iteratively take the π-positions of the blocks in $\mathcal{B}$ and compute for each block the barycenter or median of its adjacent blocks, respectively. Then, we sort $\mathcal{B}$ according to these values. The following benchmarks show that both are fast, however, they are not competitive with global sifting in the number of obtained crossings. One round of global barycenter or global median has a time complexity of $\mathcal{O}(|E| \log |E|)$ or of $\mathcal{O}(|E|)$, respectively.

We have compared the practical performance of four level-by-level and four global crossing reduction algorithms implemented in our graph tool Gravisto [5]: iterative one-sided 2-level barycenter (B), median (M), sifting (S), iterative centered 3-level sifting (3S), global barycenter (GB), global median (GM), global sifting (GS), and ordered k-level sifting (OS). For each of the level-by-level algorithms we run ten top-down and bottom-up sweeps and for each of the global heuristics we performed ten rounds. We have tested 910 random graphs. For each graph size from 1,000 to 10,000 vertices in steps of 100 we have generated ten arbitrary graphs with an aspect ratio of the *golden rectangle* $\frac{1+\sqrt{5}}{2}$, i. e., the maximum number of (dummy) vertices per level is about 1.6 times the number of levels. The density of the proper edges is twice the number of vertices including a proportion of 75% dummy vertices. Hence, there are five times as many (long) edges as non-dummy vertices. To aggregate random initializations we applied each algorithm twice to every concrete graph instance. All benchmarks were run on a 2.83 GHz XEON workstation under Solaris and the Java 6.0 platform of Oracle Corp.



Graph size $|V'|$ (75% dummy vertices and $|E'| = 2 \cdot |V'|$, i. e., $|E| = 5 \cdot |V|$)

Figure 4: Benchmark: running times

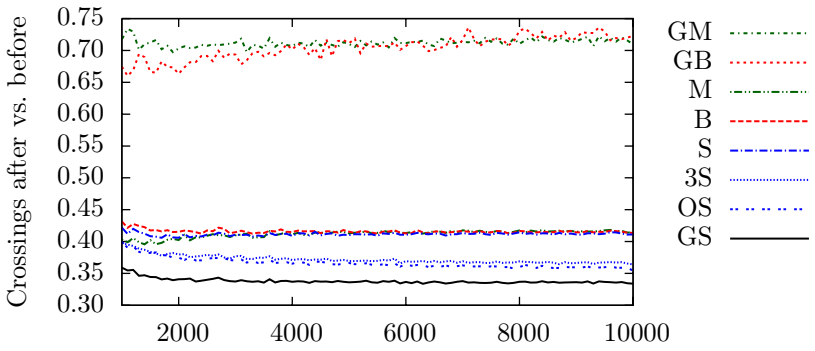
We compare the different running times in Fig. 4. Although global sifting is the slowest among them, it is feasible in practice. Clearly, its performance on real-world graphs is not as dire as the worst case $\mathcal{O}(|E|^2)$ time complexity indicates at first glance.

Fig. 5 shows the quality of the heuristics in the number of crossings of the resulting embeddings. The results of global sifting are about 5 to 10% better than the ones of the established algorithms. In addition, type 2 conflicts are avoided, which are quite high for the sifting algorithms as Fig. 6 indicates, and which have an impact on the edge routing in the final phase of the four phase framework. Both benefits justify the higher running time.

Fig. 7 depicts that the traditional approaches have a constant running time on graphs with the same size but ascending proportions of dummy vertices. For global sifting the running times become better with more dummy vertices. Then, there are more long edges whose inner segments are treated as a whole. This reflects that the time complexity $\mathcal{O}(|E|^2)$ depends only on the number of edges $|E|$ and not on the number of segments $|E'|$ of the proper graph.

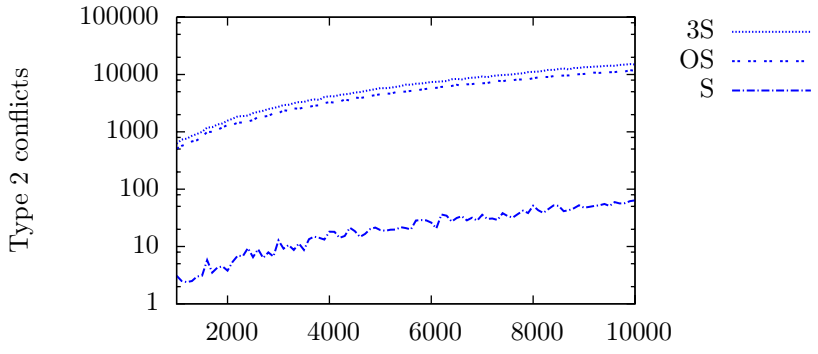
In Fig. 8 we compare the results of the heuristics with the exact solution. For a practically solvable ILP of the exact algorithm which will be described in Sect. 5.1, the graphs must be small with $|V'| \leq 35$. Although permitted, the optimum solutions do not contain any type 2 conflicts. The graphs seem to be simply too small for that. Since we have a proportion of 30% dummy vertices, the graphs are rather sparse. This may be the reason why barycenter here outperforms two sifting algorithms. Global sifting is in parts 25% closer to the optimum as all other tested methods. Fig. 5 supports the statement in [31] that in general sifting is qualitatively the better choice.

Fig. 9 presents 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 only one sweep. The remaining heuristics (OS, 3S, GS) improve their results



Graph size $|V'|$ (75% dummy vertices and $|E'| = 2 \cdot |V'|$, i. e., $|E| = 5 \cdot |V|$)

Figure 5: Benchmark: number of crossings after vs. before applying the crossing reduction



Graph size $|V'|$ (75% dummy vertices and $|E'| = 2 \cdot |V'|$, i. e., $|E| = 5 \cdot |V|$)

Figure 6: Benchmark: number of type 2 conflicts

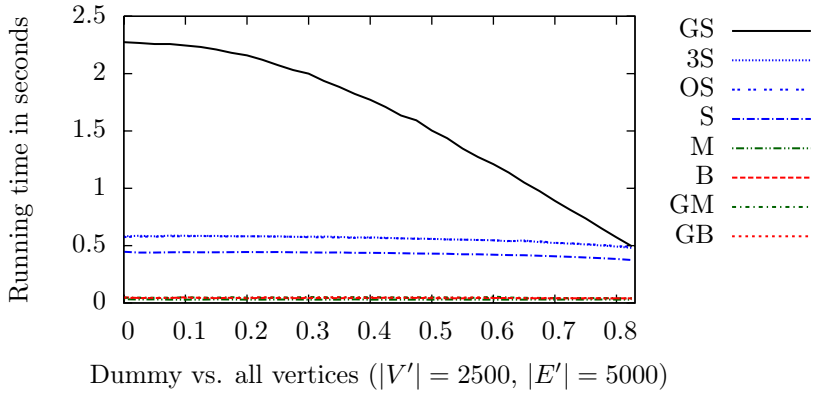


Figure 7: Benchmark: running times with different proportions of dummy vertices

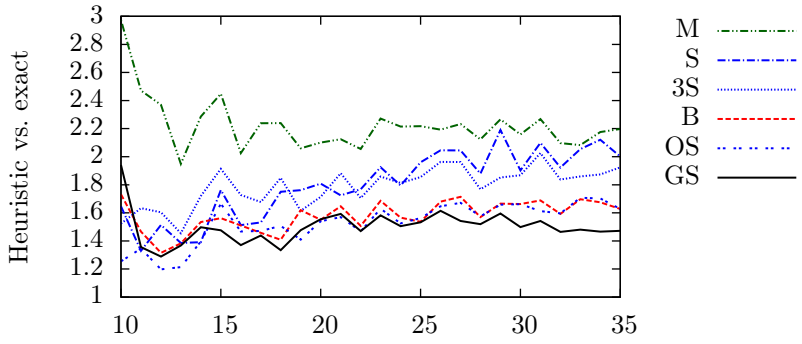


Figure 8: Benchmark: number of crossings compared to optimal solutions

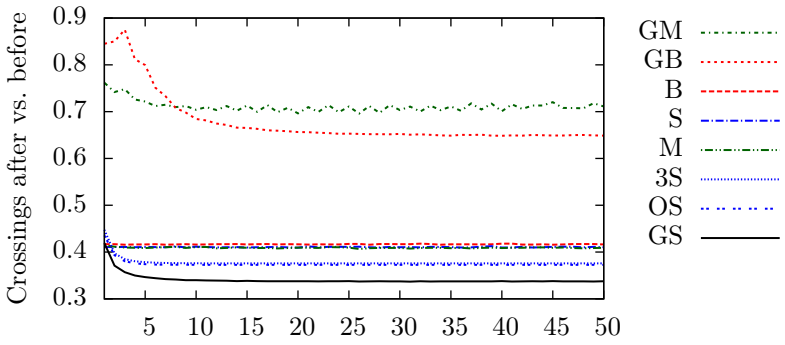


Figure 9: Benchmark: number of sweeps or rounds on graphs with 2500 (dummy) vertices

over the first five sweeps or rounds. As tests with different sizes of graphs gave similar diagrams, we conclude that at least ten rounds of global sifting should be sufficient in practice.

We additionally have tested the algorithms on the widely used Rome graphs [16] library in Fig. 10. The set contains 11528 instances with 10–100 vertices and 9–158 edges. Although these graphs are originally undirected, we interpret them as directed by artificially directing the edges according to the vertex order given in the input files, see, e. g., [12, 20]. Furthermore, we have tested the AT&T graphs [38], see Fig. 11. This benchmark set consists of 1155 directed acyclic graphs with 10–96 vertices and 10–99 edges collected by Stephen North. Since these graphs are rather inhomogeneous in the number of vertices, we follow [15] and group the graphs by the number of edges to improve the readability of the diagram. For both benchmarks we have done the leveling with the Coffman/Graham algorithm [13], where we allowed a maximum number of $\sqrt{|V|}$ non-dummy vertices on the levels. Since the graphs are rather sparse and contain many chains, we have marked all vertices with degree 2 as dummy vertices in a preprocessing step. This helps to build reasonable blocks for global sifting. In both benchmarks global sifting gives the best results of all algorithms guaranteeing no type 2 conflicts.

In a nutshell, classic sifting is fast, leaves few type 2 conflicts, but many crossings. Centered 3-level sifting is fast, leaves few crossings, but many type 2 conflicts. Global sifting leaves even fewer crossings without any type 2 conflicts, but has the highest running time which is still feasible in practice. The measurements reflect that the running time of global sifting is independent of the number of dummy vertices. This parallels the edge bundling technique in [21]. Global sifting is a good choice for the trade-off between time and quality.

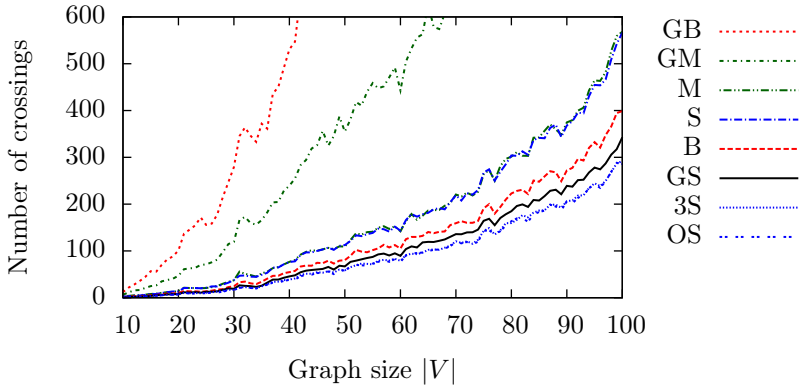


Figure 10: Benchmark: number of crossings in the Rome graphs

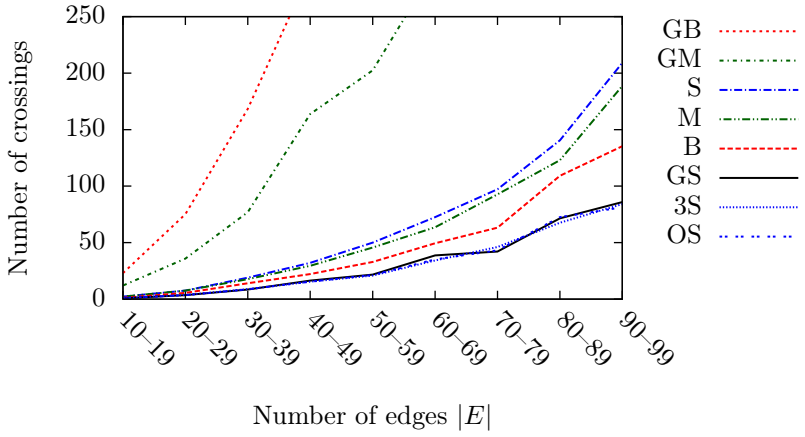


Figure 11: Benchmark: number of crossings in the AT&T graphs

5 Applications of the Global Crossing Reduction

In several related algorithms blocks for long edges can be used to improve their performance and the quality of the resulting drawings by avoiding type 2 conflicts.

5.1 Optimal Crossing Reduction Using an ILP

Jünger et al. [29] introduced an ILP formulation for the exact crossing minimization problem of k-level graphs. There, type 2 conflicts can be excluded by adding additional constraints, which after a simplification result in a similar ILP as in our approach. However, using variables for pairs of overlapping blocks with common levels gives a more direct formulation which naturally excludes type 2 conflicts. It uses fewer variables by avoiding the use of dummy vertices for computing a solution of the ILP. However, for forming the equations the graph must be proper.

Our ILP is built as follows: We start with an arbitrary but fixed order of the list of blocks $\mathcal{B}$. For any two blocks A and B with $\pi(A) < \pi(B)$ and a common level we define a boolean variable x_{AB}. The value $x_{AB} = 1$ denotes that A is left of B and $x_{AB} = 0$ that B is left of A in the final embedding. For each triple of blocks A, B, and C with $\pi(A) < \pi(B) < \pi(C)$ with 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 exclude cyclic dependencies within a level.

Let $s_1 = (a, b)$ and $s_2 = (c, d)$ be segments between the same levels such that at least one of them is an outer segment. Let $A = \text{block}(a), B = \text{block}(b), C = \text{block}(c)$, and $D = \text{block}(d)$. 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 or not 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. See (1) for the complete ILP formulation for a proper k-level graph $G' = (V', E', \phi')$ of G with $I \subset E'$ denoting the set of inner segments. Informally speaking, each element of the set $\mathcal{C}$ denotes the up to four incident blocks of each pair of edges which may cross.

$$
\begin{aligned}
\chi_{\text{opt}} = \min & \sum_{(A,B,C,D)\in\mathcal{C}} c_{ABCD} \\
\mathcal{C} = \{\, (A,B,C,D) \in \mathcal{B}^4 \mid\ & \exists (a,b),(c,d) \in E' : \\
& (a,b) \notin I \vee (c,d) \notin I, \\
& \phi(b) = \phi(d), \\
& A = \text{block}(a), B = \text{block}(b), \\
& C = \text{block}(c), D = \text{block}(d), \\
& \pi(A) < \pi(C) \,\}
\end{aligned}
\tag{1}
$$

subject to

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

According to our experiments, this approach is usable for graphs with up to forty vertices without further polyhedral studies. This is comparable with the original approach in [29].

5.2 Level Planarity Testing Using the Vertex Exchange Graph

Harrigan and Healy [27] introduced *vertex exchange graphs* as a data structure for a simple level planarity test in $\mathcal{O}(|V'|^2)$ time, where $V' \supseteq V$ is the set of all vertices including dummy vertices. Using our blocks, it is straightforward to improve the running time of the test from $\mathcal{O}(|V'|^2)$, 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 for testing level planarity or to minimize crossings without type 2 conflicts [39].

5.3 Clustered Crossing Reduction

In a clustered level graph the vertices are grouped into subgraphs in a hierarchical way. The crossing reduction has to ensure that all (dummy) vertices of a subgraph in the same level are consecutive and that all subgraphs spanning several levels have a matching order in each level that avoids overlaps of disjoint subgraphs, see [23, 24]. The treatment of directed clustered graphs is rather complicated using a 2-level crossing reduction approach. With global sifting we can address crossings directly. Instead of swapping a vertex with its right neighbor in a sifting swap we swap all blocks of a subgraph with its right neighbor, which itself is either a block or a subgraph, and determine the change in the number of crossings. The time complexity stays the same as in the ordinary global sifting algorithm. 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 every hierarchical layer.

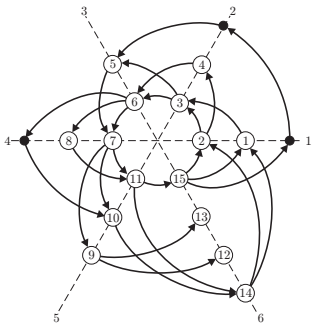
5.4 Cyclic and Radial Level Graphs

Level graphs can be generalized to cyclic and to radial level graphs. In cyclic level graphs the set of levels is ordered in a cyclic way, i. e., the first level directly succeeds the last. This corresponds to the *recurrent hierarchies* of Sugiyama et al. [43]. Cyclic levels are normally drawn forming a star in 2D (see Fig. 12(a)). These drawings explicitly visualize cycles in graphs [6], which is helpful, e. g., in scheduling [26, 41, 44] and in bioinformatics [35]. In radial level graphs each level itself is ordered in a cyclic way, i. e., the first vertex in each level is the right neighbor of the last vertex. The levels are drawn as concentric rings. See Figs. 12 and 13 for example drawings. Global sifting can be extended to both concepts and is the first crossing reduction technique avoiding type 2 conflicts in each case.

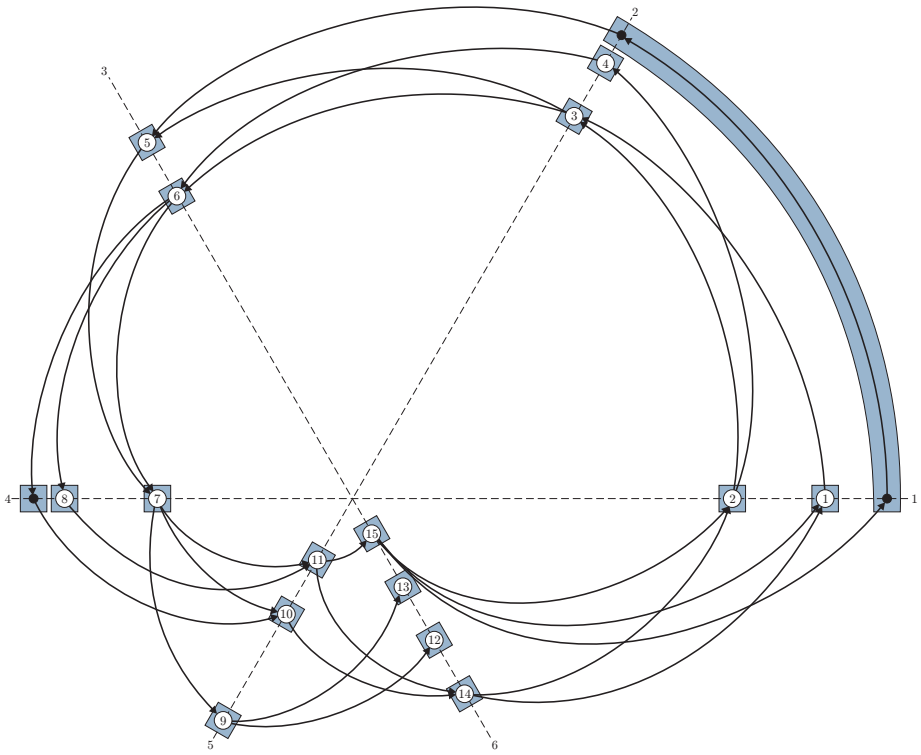
For cyclic level graphs our global sifting can be used without any changes and within the same time complexity. Note that one-sided 2-level algorithms cannot be applied here, since each of them pushes most of the crossings to the next level and these form a cycle. Even the absence of type 2 conflicts cannot be guaranteed then, because the sweep has to stop at some level. There is no possibility to push the crossings and the type 2 conflicts upwards or downwards out of the drawings as it is in usual hierarchical level drawings. The conflicts move like a wave front from level to level and they recur.

For leveling and coordinate assignment of cyclic level graphs see [2, 4]. The ILP approach by Jünger et al. [29] can be used for exact cyclic k-level crossing minimization straightforward by including additional constraints for the edges between the last and the first level.

In a radial level graph the levels are concentric circles (see Fig. 13(a)). These drawings visualize distance or importance, and are the common drawings of social networks [9,46]. They map structural centrality of the graph to geometric centrality. Crossing minimization in radial level graphs is $\mathcal{NP}$-hard, even if restricted to two levels with one side fixed [1]. 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 order of $\mathcal{B}$ starts with an arbitrary block. As in [1] we define an *offset* $\psi : E \to \mathbb{Z}$ for each outer segment. The absolute value $|\psi(e)|$ counts the crossings of segment e with an imaginary *ray* splitting the levels by a straight halfline from the concentric center to infinity. If $\psi(e) < 0$ ($\psi(e) > 0$), e has clockwise (counter-clockwise) direction from the source to the target. When sifting a block $A \in \mathcal{B}$, we have to update the *partings*, which are the two borders between the counterclockwise and clockwise segments in the levels above and below A, see Fig. 13(b). Since we can do this independently of each other and add the results of the change in crossings to Δ in Algorithm 4, we use the same technique as in [1]. We sift a block from its current position in counterclockwise direction. Thus, for few crossings the partings have to follow this direction in their levels. During the swap the test whether or not changing the orientation of some of the first of the (ordered) incident segments of A by incrementing their offsets, and thus putting them last, leads to less crossings. However, counting the difference raises the

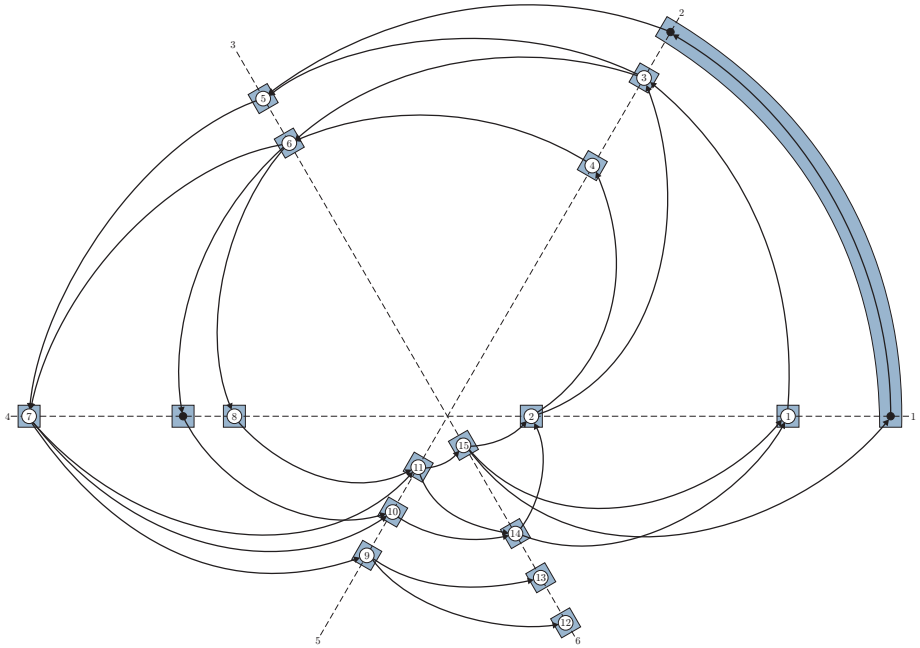


(a) Input

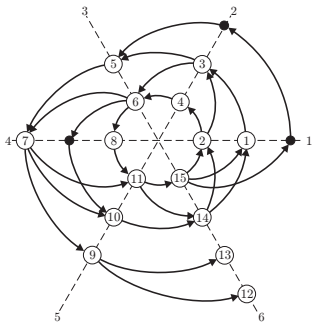


(b) Unique radii for the blocks

Figure 12: Global crossing reduction for a cyclic drawing



(c) After global sifting



(d) Resulting Embedding

Figure 12: Global crossing reduction for a cyclic drawing (cont.)

overall running time to $\mathcal{O}(|E|^3)$. The radial coordinate assignment phase in [1] relies on the obtained absence of type 2 conflicts.

6 Discussion

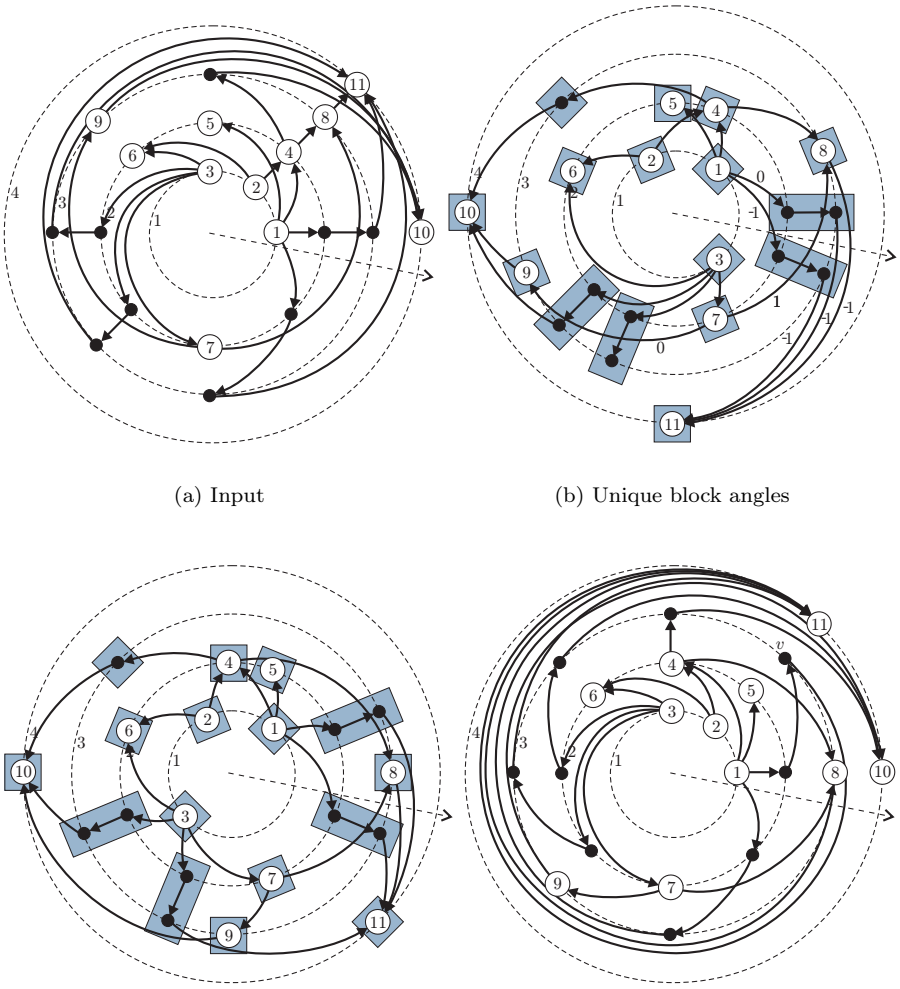
We have presented an algorithm for the global crossing reduction problem of k-level graphs. It produces high quality results with fewer crossings than common approaches at the expense of a quadratic running time which is still feasible in practice. Global crossing reduction was an open problem since the introduction of the hierarchical framework [43] in 1981. For radial and cyclic level crossing reduction our algorithm is the first which guarantees the absence of type 2 conflicts. Our approach simplifies and improves several other algorithms concerning level planarity and crossing reduction.

6.1 Open Problems

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. 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 conceptually 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.

A first approach to applying the blocks to the radial crossing reduction algorithm in [1] 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.



(a) Input

(b) Unique block angles

(c) After global sifting

(d) Resulting Embedding

Figure 13: Global crossing reduction for a radial drawing

References

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

[2] 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.

[3] 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.

[4] C. Bachmaier, F. J. Brandenburg, W. Brunner, and G. Lovász. Cyclic leveling of directed graphs. In I. G. Tollis and M. Patrignani, editors, *Proc. Graph Drawing, GD 2008*, volume 5417 of *LNCS*, pages 348–349. Springer, 2009.

[5] 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.

[6] C. Bachmaier and W. Brunner. Linear time planarity testing and embedding of strongly connected cyclic level graphs. In D. Halperin and K. Mehlhorn, editors, *ESA 2008*, volume 5193 of *LNCS*, pages 136–147. Springer, 2008.

[7] C. Bachmaier, W. Brunner, and A. Gleißner. Grid sifting: Leveling and crossing reduction. Submitted for publication.

[8] 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.

[9] U. Brandes and T. Erlebach, editors. *Network Analysis: Methodological Foundations*, volume 3418 of *LNCS Tutorial*. Springer, 2005.

[10] U. Brandes and B. Köpf. Fast and simple horizontal coordinate assignment. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Proc. Graph Drawing, GD 2001*, volume 2265 of *LNCS*, pages 31–44. Springer, 2002.

[11] M. Chimani, C. Gutwenger, P. Mutzel, and H.-M. Wong. Layer-free upward crossing minimization. *ACM J. Exp. Alg.*, 15:2.2.1–2.2.27, 2010.

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

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

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

[15] G. Di Battista, A. Garg, G. Liotta, A. Parise, R. Tamassia, E. Tassinari, F. Vargiu, and L. Vismara. Drawing directed acyclic graphs: An experimental study. *Internat. J. Comput. Geom. Appl.*, 10(6):623–648, 2000.

[16] G. Di Battista, A. Garg, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of four graph drawing algorithms. *Comput. Geom. Theory Appl.*, 7(5–6):303–325, 1997.

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

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

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

[20] M. Eiglsperger, F. Eppinger, and M. Kaufmann. An approach for mixed upward planarization. *J. Graph Alg. App.*, 7(2):203–220, 2003.

[21] M. Eiglsperger, M. Siebenhaller, and M. Kaufmann. An efficient implementation of sugiyama's algorithm for layered graph drawing. *J. Graph Alg. App.*, 9(3):305–325, 2005.

[22] T. Eschbach, W. Günther, R. Drexler, and B. Becker. Crossing reduction by windows optimization. In M. T. Goodrich and S. G. Kobourov, editors, *Proc. Graph Drawing, GD 2002*, volume 2528 of *LNCS*, pages 285–294. Springer, 2002.

[23] M. Forster. *Crossings in Clustered Level Graphs*. PhD thesis, University of Passau, 2004.

[24] M. Forster and C. Bachmaier. Clustered level planarity. In P. Van Emde Boas, J. Pokorný, M. Bieliková, and J. Štuller, editors, *Proc. Software Seminar: Theory and Practice of Informatics, SOFSEM 2004*, volume 2932 of *LNCS*, pages 218–228. Springer, 2004.

[25] M. R. Garey and D. S. Johnson. Crossing number is $\mathcal{NP}$-complete. *SIAM J. Alg. Discr. Meth.*, 4(3):312–316, 1983.

[26] 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.

[27] 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.

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

[29] 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.

[30] M. Jünger and P. Mutzel. 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. *J. Graph Alg. App.*, 1(1):1–25, 1997.

[31] M. Kaufmann and D. Wagner. *Drawing Graphs*, volume 2025 of *LNCS*. Springer, 2001.

[32] P. Kuntz, B. Pinaud, and R. Lehn. Minimizing crossings in hierarchical digraphs with a hybridized genetic algorithm. *J. Heuristics*, 12(1–2):23–26, 2006.

[33] M. Laguna, R. Martí, and V. Valls. Arc crossing minimization in hierarchical digraphs with tabu search. *Comp. Oper. Res.*, 24(12):1175–1186, 1997.

[34] 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.

[35] G. Michal, editor. *Biochemical Pathways: An Atlas of Biochemistry and Molecular Biology*. Wiley, 1998.

[36] X. Muñoz, W. Unger, and I. Vrt'o. One sided crossing minimization is $\mathcal{NP}$-hard for sparse graphs. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Proc. Graph Drawing, GD 2001*, volume 2265 of *LNCS*, pages 115–122. Springer, 2002.

[37] H. Nagamochi. An improved bound on the one-sided minimum crossing number in two-layered drawing. *Discrete Comput. Geom.*, 33:569–591, 2005.

[38] S. North. AT&T Graph Collection. `http://www.graphdrawing.org/`.

[39] 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. *Elect. Notes in Discr. Math.*, 9:269–277, 2001.

[40] 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.

[41] P. Serafini and W. Ukovich. A mathematical model for periodic scheduling problems. *SIAM J. Discr. Math.*, 2(4):550–581, 1989.

[42] M. Stallmann, F. Brglez, and D. Ghosh. Heuristics, experimental subjects, and treatment evaluation in bigraph crossing minimization. *ACM J. Exp. Alg.*, 6(8), 2001.

[43] 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.

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

[45] J. Utech, J. Branke, H. Schmeck, and P. Eades. An evolutionary algorithm for drawing directed graphs. In H. R. Arabnia, editor, *Proc. International Conference on Imaging Science, Systems, and Technology, CISST 1998*, pages 154–160. CSREA, 1998.

[46] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications (Structural Analysis in the Social Sciences).* Cambridge University Press, 1994.