A Global k-Level Crossing Reduction Algorithm*

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

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

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

1 Introduction

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

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

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

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

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

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

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

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

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

2 Preliminaries

We suppose that a directed graph without self-loops has passed through the cycle removal and leveling phases. The outcome is a *k-level graph* $G = (V, E, \phi)$, where $\phi\colon V \to \{1, 2, \ldots, k\}$ is a surjective level assignment of the vertices with $\phi(u) < \phi(v)$ for each edge $(u, v) \in E$. For an edge $e = (u, v) \in E$ we define $\text{span}(e) := \phi(v) - \phi(u)$. 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 [4], short edges are called *segments* of e. The first and the last segments are the *outer* and the others the *inner segments*. Inner segments connect two dummy vertices.

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

Next we define blocks, which prevent dealing with dummy vertices and so keep the running time independent of them. A *block* is a single vertex of V or a maximum connected subgraph of dummy vertices, i. e., the inner segments of a long edge. The blocks represent the vertices of a graph related to G', where the edges are the outer segments. For a block A define $x = \text{upper}(A)$ ($y = \text{lower}(A)$) 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. We define $N^-(A) := N^-(\text{upper}(A))$, $N^+(A) := N^+(\text{lower}(A))$, $\deg(A) := |N^-(A)| + |N^+(A)|$, 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 $v \in V'$. Let $\mathcal{B}$ be any ordered list of all blocks and let $\pi\colon \mathcal{B} \to \{0, \ldots, |\mathcal{B}| - 1\}$ assign each block its current *position* in this ordering.

3 Global Sifting

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

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

Algorithm 1: GLOBAL-SIFTING

Input: Proper k-level graph $G' = (V', E', \phi')$, number 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 return G'
```

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

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

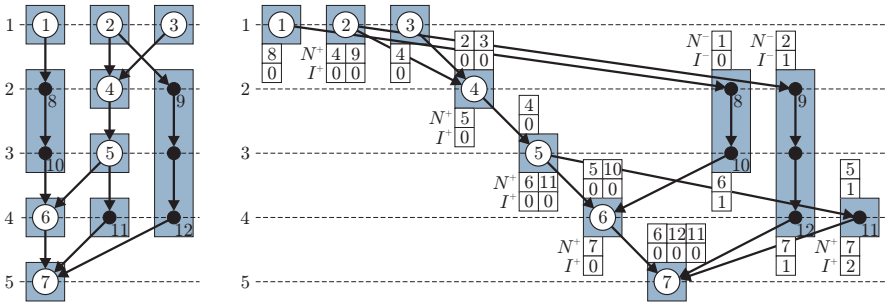
3.1 Building the Block List

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

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

3.2 Initialization of a Sifting Step

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



(a) A level graph with ten blocks

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

Fig. 1. Blocks as sifting objects

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

Algorithm 2: SORT-ADJACENCIES

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

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

3.3 Sifting Step

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

Algorithm 3: SIFTING-STEP

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

```
1 B' ← A ≺ B[0] ≺ ··· ≺ B[|B| − 1]      // new ordering 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 is the actual computation of the change in the number of crossings when a block A is swapped with its right neighbor B. In contrast to one-sided crossing reduction, our global approach takes the whole neighborhood of both blocks into account when the change in the number of crossings is computed. Lemma 1 states which segments are involved.

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

Proof. Note that only segments between the same levels can cross. As no type 2 conflicts occur at least one of the segments of a crossing has to be an outer segment. Let (a, b) and (c, d) be two segments between the same levels with $a \neq c$ and $b \neq d$. If the two segments cross after swapping A and B but did not cross before (or vice versa) either a and c or b and d were swapped. Therefore, one of

the segments is adjacent to A or is a part of A and the other is adjacent to B or is a part of B. If b and d were swapped and thus a and c were not, $\phi(b) = \phi(d)$ is the upper level of A or B and thus one of the crossing segments is an incoming outer segment of A or B. The other segment is either an incoming outer segment or an inner segment of the other block. Note that it cannot be an outgoing outer segment of this block because then neither a and c nor b and d would have been swapped. The other case of swapping a and c instead of b and d is symmetric. □

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

Algorithm 4 shows the details of a sifting swap. First, the levels at which (significant) swaps occur and the direction of the segments changing their crossings are found (lines 2–6). For each entry (l, d) of the set $\mathcal{L}$ the two vertices a and b of A and B on level l are retrieved. Note that when swapping A and B only a and b are swapped on their level and that in the level of their neighbors $N^d(a)$ and $N^d(b)$ no order changes. Thus, the computation of the change in the number of crossings can be done as in [3] (lines 13–23): The neighbors are traversed from left to right. If a neighbor of a is found (lines 18, 19) its segment will cross all remaining $s - j$ incident segments of b after the swap. If a neighbor of b is found (lines 20, 21) 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 22). An update of the adjacency after a swap (line 10) is only necessary if a and b have common neighbors. Algorithm 5 shows how this can be done in overall $\mathcal{O}(\deg(A) + \deg(B))$ time similarly to the crossing counting function uswap.

3.5 Time Complexity

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

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

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

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

Algorithm 4: SIFTING-SWAP

Input: Consecutive blocks A, B
Output: Change in crossing count

begin
  $\mathcal{L} \leftarrow \emptyset; \Delta \leftarrow 0$
  if $\phi(\text{upper}(A)) \in \text{levels}(B)$ **then** $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\phi(\text{upper}(A), -)\}$
  if $\phi(\text{lower}(A)) \in \text{levels}(B)$ **then** $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\phi(\text{lower}(A), +)\}$
  if $\phi(\text{upper}(B)) \in \text{levels}(A)$ **then** $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\phi(\text{upper}(B), -)\}$
  if $\phi(\text{lower}(B)) \in \text{levels}(A)$ **then** $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\phi(\text{lower}(B), +)\}$
  foreach $(l, d) \in \mathcal{L}$ **do**
    let a in A and b in B be the vertices with $\phi(a) = \phi(b) = l$
    $\Delta \leftarrow \Delta + \mathsf{uswap}(a, b, N^d(a), N^d(b))$
    UPDATE-ADJACENCY$(a, b, N^d(a), I^d(a), N^d(b), I^d(b))$
  swap positions of A and B in $\mathcal{B}$; $\pi(A) \leftarrow \pi(A) + 1$; $\pi(B) \leftarrow \pi(B) - 1$
  return Δ

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

Algorithm 5: UPDATE-ADJACENCIES

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

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

4 Simple Global Crossing Reductions

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

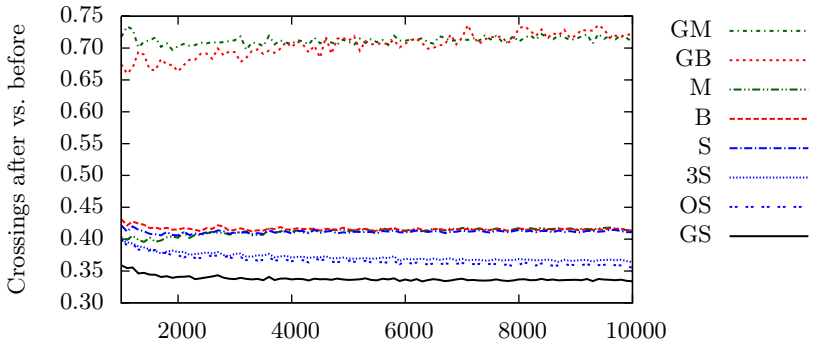
Theorem 2. *One round of global barycenter or global median has time complexity* $\mathcal{O}(|E| \log |E|)$ *or* $\mathcal{O}(|E|)$*, respectively.*

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

5 Experimental Results

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

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



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

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

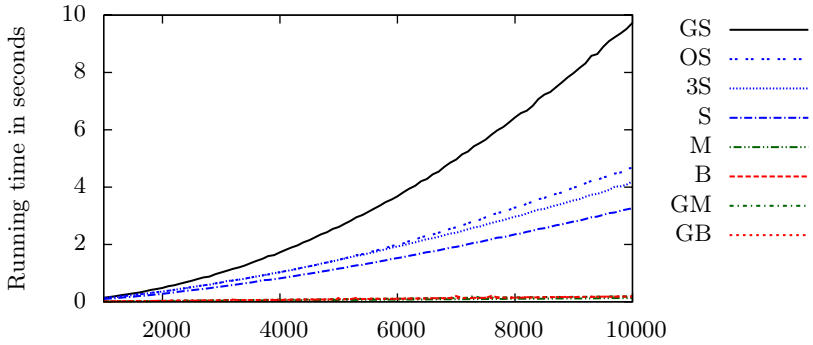


Fig. 3. Benchmark: running times

6 Applications of the Global Crossing Reduction

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

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

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

Cyclic and Radial Level Graphs Level graphs can be extended to cyclic or radial level graphs. In cyclic level graphs the set of levels is ordered in a cyclic way, i. e., the first level follows the last one. In radial level graphs each level itself is ordered in a cyclic way, i. e., the first vertex on each level is the right neighbor

of the last one. See Fig. 4 for clippings of drawings. For both, global sifting is the first crossing reduction to guarantee the needed absence of type 2 conflicts.

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

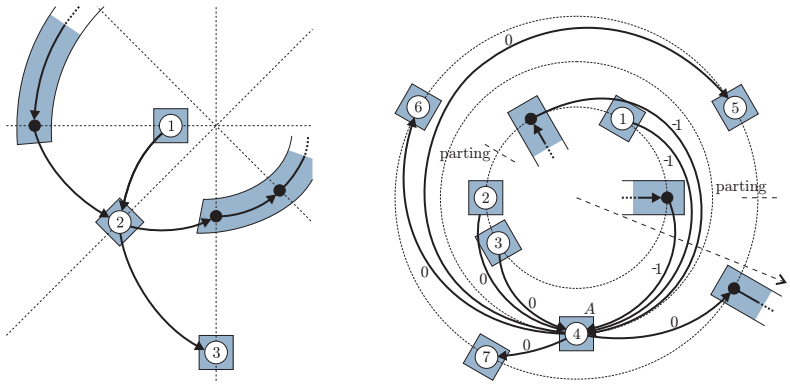
In a radial level graph the levels are concentric circles (see Fig. 4(b)). These drawings visualize distances or importance and are the traditional drawings of social networks. They map structural centrality of the graph to geometric centrality. Our global sifting approach guarantees radially aligned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

7 Summary

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

References

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



(a) Cyclic drawing

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

Fig. 4. Clippings of cyclic and radial drawings

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