Grid Sifting: Leveling and Crossing Reduction

Christian Bachmaier, Wolfgang Brunner, Andreas Gleißner

Department of Informatics and Mathematics, University of Passau
{bachmaier,brunner,gleissner}@fim.uni-passau.de

Technical Report, Number MIP-1103
Department of Informatics and Mathematics
University of Passau, Germany
February 2011

Grid Sifting: Leveling and Crossing Reduction*

Technical Report MIP-1103
February, 2011

Christian Bachmaier, Wolfgang Brunner, and Andreas Gleißner

University of Passau
94030 Passau, Germany
{bachmaier,brunner,gleissner}@fim.uni-passau.de

Abstract. Directed graphs are commonly drawn by the Sugiyama algorithm where first vertices are placed on distinct hierarchical levels and second the vertices on the same level are permuted to reduce the overall number of crossings. Separating these two phases simplifies the algorithms but diminishes the quality of the result.
We introduce a combined leveling and crossing reduction algorithm based on sifting, which prioritizes few crossings over few levels. It avoids type 2 conflicts which help to straighten the edges, and has a running time, which is roughly quadratic in the size of the input graph independent of dummy vertices.

1 Introduction

The Sugiyama framework [16] 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 according to some aesthetic criteria). Typical applications are schedules, UML diagrams, and flow charts.

There are many different leveling and crossing reduction algorithms. Traditional leveling methods minimize the number of levels by the longest path method [13], by the Coffman/Graham algorithm with a predefined maximum width [8], or by Gansner et al.'s ILP minimizing the sum of the edge lengths [11]. The common solution for *k-level crossing minimization* is a reduction to the still $\mathcal{NP}$-hard [10] *one-sided 2-level crossing minimization problem*, which is repeatedly solved by some up and down sweeps [13, 16]. Bastert and Matuszewski claim [13] 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" [13, page 102].

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

An important feature of crossing reduction algorithms is the avoidance of *type 2 conflicts*, which are crossings of two edges between dummy vertices. Among others, the standard fourth phase algorithm [5] by Brandes and Köpf assumes the absence of type 2 conflicts. Then it aligns long edges vertically and so achieves a crucial aesthetic criterion [13] regarding pleasing hierarchical drawings.

Sifting was at first used for vertex minimization in ordered binary decision diagrams [15] and was later on adapted to the one-sided 2-level crossing reduction [14]. The idea is to keep track of the number of crossings while in a *sifting step* a vertex v on level i is moved along a fixed order of all the vertices V_i on level i. Finally v is placed at its locally optimal position. We call a single swap a *sifting swap* and the execution of a sifting step for every vertex in V_i a *sifting round*.

Matuszewski et al. [14] have extended sifting towards a global view. There the vertices of all levels are sorted by their degree and are first sifted in increasing order and then in decreasing order. Jünger et al. [12] 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 suggested in literature, e. g., genetic algorithms, tabu search, or windows optimization. We have presented a global sifting algorithm [1], which aligns long edges to blocks and tries to find an apt position for each block as a whole.

Although these algorithms try to achieve a global view on the overall number of crossings, they do not change the leveling. "Since level assignment and crossing reduction are realized as independent steps, the resulting drawings may have unnecessary crossings caused by an unfortunate level assignment" [7, p. 94]. A minimal graph showing this phenomenon is the $K_{2,2}$. With two vertices on level 1 and two on level 2 the graph has one crossing. With one vertex on a third level the crossing can be eliminated. In general, more levels lead to fewer crossings.

Recently, Chimani et al. [6, 7] presented a planarization approach to reduce the number of crossings. They compute a planar upward embedding of the graph by adding dummy vertices where two edges would cross. This embedding is leveled respecting the given orders of the adjacency lists. Then the number of at most $\mathcal{O}(|V|)$ levels is reduced by compacting the drawing. The algorithm needs $\mathcal{O}(|E|^5)$ time and still $\mathcal{O}(|E|^3) = \mathcal{O}(|V|^3)$ time for planar input graphs.

In this paper we propose a new leveling and crossing reduction technique which combines both phases in a natural way. We extend global sifting [1] such that the blocks are not only sifted "horizontally" on their levels but also "vertically" on all suitable levels and prioritize the crossing reduction over the leveling. The algorithm yields better results than traditional heuristics. It avoids type 2 conflicts and runs in roughly quadratic time in the size of the input graph.

2 Preliminaries

We suppose that a directed graph without self-loops has passed through the cycle removal phase and has been assigned an initial leveling. 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.

The traditional approach is to make each level graph 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 [5], we call short edges *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 *(level) embedded* proper level graph, the vertices on each level are ordered from left to right. In an embedded level graph two segments are *conflicting* 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.

Similarly to [5], a *block* either represents a vertex of V, i. e., a *vertex block*, or an edge of E, i. e., an *edge block*. A block is an entity of its own and is not treated as a set. The sets of vertex blocks, edge blocks, and all blocks are called $\mathcal{B}_V$, $\mathcal{B}_E$, and $\mathcal{B} = \mathcal{B}_V \dot\cup \mathcal{B}_E$. For each $x \in V \cup E$ denote by $\text{block}(x)$ the block representing x. The set $\mathcal{F} := \{(\text{block}(u), \text{block}(e)) \in \mathcal{B}_V \times \mathcal{B}_E : e = (u, \cdot)\} \cup \{(\text{block}(e), \text{block}(v)) \in \mathcal{B}_E \times \mathcal{B}_V : e = (\cdot, v)\}$ models a relation between blocks induced by the incidence relation in G. $\mathcal{G} := (\mathcal{B}, \mathcal{F})$ forms the directed *block graph* of G. For a long edge e, its edge block $\text{block}(e)$ corresponds to the maximum connected subgraph of dummy vertices in G', i. e., the inner segments of e. Such blocks are called *active*. Likewise, blocks corresponding to short edges, which do not need any dummy vertices in G', are called *inactive*. Edges of the block graph that are incident to an active edge block correspond to outer segments of G'. For a dummy vertex v' in G' let $\text{block}(v')$ be the edge block corresponding to the subgraph containing v'.

Note that G' and the notion of dummy vertices are only used to simplify definitions and are never directly respresented in the algorithm. As the grid sifting algorithm modifies the level assignment ϕ of vertex blocks, edge spans as well as the number of dummy vertices of G' change over time. Hence, edge blocks switch between the active and inactive state.

Due to technical reasons explained later, most vertex blocks lie on even levels. If an odd level number is not assigned to any vertex block, the level is *nonexistent*. In G' there is no need to create dummy vertices for that level number. Let $\text{next}^+(l)$ (resp. $\text{next}^-(l)$) be the next existent level after (before) level l. A block is defined to *use* a level if a (dummy) vertex of the corresponding subgraph in G' is assigned to this level. Let $\text{levels}(B)$ be the set of all levels used by block B. We define $\overline{\phi}(B) := \min \text{levels}(B)$ (resp. $\underline{\phi}(B) := \max \text{levels}(B)$) to be the topmost (undermost) level used by B. For each vertex block $B = \text{block}(v)$ $\overline{\phi}(B) = \underline{\phi}(B) = \phi(B) := \phi(v)$ holds. An active edge block $B = \text{block}((u, v))$ uses the levels $\overline{\phi}(B) = \text{next}^+(\phi(u)), \text{next}^+(\text{next}^+(\phi(u))), \ldots, \text{next}^-(\phi(v)) = \underline{\phi}(B)$. For inactive edge blocks B, the set $\text{levels}(B)$ is empty.

We introduce l-neighbors of (especially edge) blocks to address parts of these blocks as if the dummy vertices were existent. If (u, v) is any segment in G' and $l = \phi(u)$, we call $\text{block}(v)$ to be an *l-neighbor* of $\text{block}(u)$ in direction $+$. Likewise we call $\text{block}(u)$ to be an *$l+1$-neighbor* of $\text{block}(v)$ in direction $-$. Note

that if additionally $\text{block}(u), \text{block}(v) \in \mathcal{B}_V$, then they are l-neighbors of each other ignoring the inactive edge block in between. For each block B, direction $d \in \{+, -\}$ and level $l \in \text{levels}(B)$ we define $N^d(B, l)$ to be the set of all l-neighbors of B in direction d. For an edge block B, $N^d(B, l)$ contains exactly one element. Note that if $\overline{\phi}(B) < l < \underline{\phi}(B)$, then $N^d(B, l) = B$, so that an edge block can be its own l-neighbor. For the perspective using dummy vertices, let v be the dummy vertex of an edge block B on level l. Then the l-neighbor of B in direction d is the block of the neighbor of v in direction d, which is either B or an adjacent vertex block. For each vertex block B we define $N^d(B) := N^d(B, \phi(B))$ as B uses only one level. 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. Note that the drawing of a short edge (u, v) is independent of $\pi(\text{block}((u, v)))$. We call the pair $\mathcal{E} = (\pi, \phi)$ a *level embedding* of the graph.

See Fig. 1(a) for an example of a graph with 5 vertices and 6 edges. Each vertex is represented by a vertex block and 5 edge blocks are shown, which are active. The edge $(2, 3)$ is short and, hence, its edge block is inactive. The levels 5 and 7 are nonexistent (dashed). Thus, $\text{next}^+(4) = 6$ and $\text{next}^-(8) = 6$. Then $\text{levels}(J) = \{3, 4, 6\}, \overline{\phi}(J) = 3, \underline{\phi}(J) = 6, N^+(J, 3) = J, N^+(J, 4) = J$ and $N^+(J, 6) = \text{block}(5) = H$. Furthermore, $N^+(D) = (B, F)$ in Fig. 1(a) (ignoring the currently inactive edge block $L = \text{block}(2, 3)$) and $N^+(D) = (L, G)$ in Fig. 1(c) (as level 3 is nonexistent, $F = \text{block}(2, 4)$ is inactive).

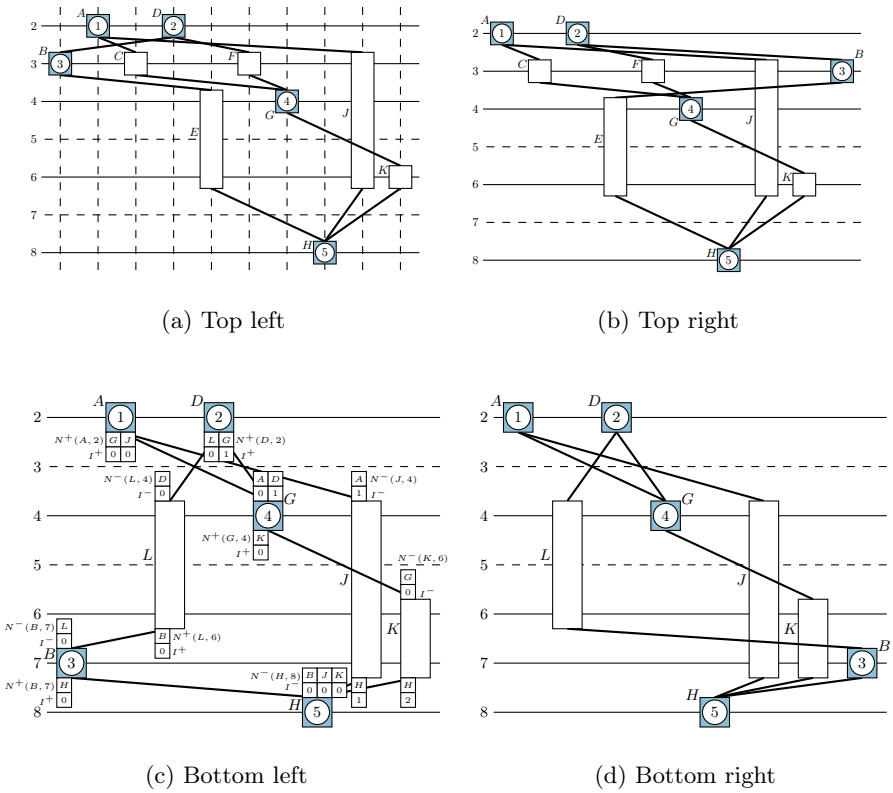
3 Grid Sifting

We extend our approach of global sifting [1] and try to find optimal positions for each (vertex) block on all levels. We use a two-dimensional grid which we place each vertex block on, see Fig. 1(a). The edge blocks span the levels between their incident vertex blocks and represent the dummy vertices of each edge. Each block has a unique x-coordinate in the grid which is given by the order of $\mathcal{B}$. As an initialization we use an arbitrary leveling and a random permutation of $\mathcal{B}$.

Finding an optimal position for a vertex block consists of two nested loops, cf. Algorithm 1. The outer loop, called *vertical step*, iterates over all levels the vertex can use without reversing an incident edge. In the inner loop, called *horizontal step*, all positions of the vertex block on the current level are tested. In the end, the vertex block is placed on the level and position causing the minimal number of crossings. If there are several positions causing the same number of crossings, we place the vertex block at the level nearest to the middle level and use the x-coordinate minimizing the horizontal edge lengths, which is omitted in pseudocode. Edge blocks are sifted horizontally only, as their levels are determined by the levels of their adjacent vertex blocks.

3.1 Vertical Sifting Step

In a vertical sifting step we test a vertex block $B \in \mathcal{B}$ at each possible level. To improve the result, we also try to place each vertex block between existing



(a) Top left (b) Top right

(c) Bottom left (d) Bottom right

Fig. 1. Extremal positions of vertex block B during its vertical step

levels. Therefore, we normalize, i. e., relabel the existing levels to $2, 4, 6, \ldots$ (line 1 in Algorithm 2) and then test the vertex block for all level numbers.

To avoid reversing edges, the uppermost possible level $l_{\min}$ for a vertex block is one level below its undermost preceding vertex block (line 2) or 1 if there is no such block. Likewise, the undermost possible level $l_{\max}$ is determined (line 3). See Fig. 1, where block B is tested with $l_{\min} = 3$ and $l_{\max} = 7$. In practice, we confine the number of tested levels above and below the old level of B, i. e., the *level radius* by a constant to improve the running time.

The algorithm does not count the total number of crossings, but only the relative differences between the levels tested. Thus, it first places B at the uppermost possible level using VERTICAL-JUMP (line 4). All crossing numbers (χ) are relative to the number of crossings produced by placing B at the leftmost position on this level. Then, step-by-step, B is swapped downwards to lower levels with VERTICAL-SWAP (Algorithm 3) and the locally calculated relative differences in crossing count Δ are summed up in χ (lines 6–8).

Algorithm 1: GRID-SIFTING

Input: Block graph $\mathcal{G} = (\mathcal{B} = \mathcal{B}_V \cup \mathcal{B}_E, \mathcal{F})$ with initial level assignment $\phi : \mathcal{B}_V \to \mathbb{Z}$, number ρ of sifting rounds
Output: Block graph $\mathcal{G}$ with embedding $\mathcal{E}$ given by blocks ordered by values $\pi(B)$ for each $B \in \mathcal{B}$ and updated level assignment ϕ

1 initialize $\pi : \mathcal{B} \to \{1, \dots, |\mathcal{B}|\}$ with random permutation
2 $\mathcal{E} \leftarrow (\phi, \pi)$
3 **for** $1 \le i \le \rho$ **do**
4 $\quad$ **foreach** $B \in \mathcal{B}_V$ **do** $\mathcal{E} \leftarrow$ `VERTICAL-STEP`$(\mathcal{G}, \mathcal{E}, B)$
5 **return** $(\mathcal{G}, \mathcal{E})$

Algorithm 2: VERTICAL-STEP$(\mathcal{G}, \mathcal{E}, B)$

Input: Block graph $\mathcal{G} = (\mathcal{B}, \mathcal{F})$, embedding $\mathcal{E} = (\phi, \pi)$ with k used levels, vertex block B to sift
Output: Updated embedding $\mathcal{E} = (\phi, \pi)$

1 normalize level numbers ϕ to $2, 4, 6, \dots, 2k$
2 **if** $N^-(B) = \emptyset$ **then** $l_{\min} \leftarrow 1$ **else** $l_{\min} \leftarrow \max_{A \in N^-(B)} \phi(N^-(A, \overline{\phi}(A))) + 1$
3 **if** $N^+(B) = \emptyset$ **then** $l_{\max} \leftarrow 2k + 1$ **else** $l_{\max} \leftarrow \min_{C \in N^+(B)} \phi(N^+(C, \underline{\phi}(C))) - 1$
4 $(\mathcal{E}, \Delta) \leftarrow$ `VERTICAL-JUMP`$(\mathcal{G}, \mathcal{E}, B, l_{\min})$
5 $\mathcal{E}_{\text{best}} \leftarrow \mathcal{E}; \chi_{\text{best}} \leftarrow \Delta; \chi \leftarrow \Delta$
6 **for** $l \leftarrow l_{\min} + 1$ **to** $l_{\max}$ **do**
7 $\quad$ $(\mathcal{E}, \Delta) \leftarrow$ `VERTICAL-SWAP`$(\mathcal{G}, \mathcal{E}, B, l)$; $\chi \leftarrow \chi + \Delta$
8 $\quad$ **if** $\chi < \chi_{\text{best}}$ **then** $\mathcal{E}_{\text{best}} \leftarrow \mathcal{E}; \chi_{\text{best}} \leftarrow \chi$
9 **return** $\mathcal{E}_{\text{best}}$

3.2 Vertical Jump and Vertical Swap

In the vertical swap, we move a vertex block B from level $l-1$ to level l (line 4 in Algorithm 3). Then we sift each of the edge blocks incident to B (lines 9–10) and B itself horizontally (line 11) and return the best embedding $\mathcal{E}$ and the sum of crossing changes Δ.

As $\phi(B)$ is modified, previously inactive edge blocks may become active. $\pi(A)$ of such an edge block A has previously been meaningless and may be outdated, as A did not correspond to any dummy vertex. To keep the number of crossings, we put each edge block becoming active at the horizontal barycenter of its adjacent vertex blocks. The injectivity of π is repaired by bucket-sort. In Figs. 1(a) and (b) the edge block block$((2,3))$ is inactive and block$((3,5))$ is active. In Figs. 1(c) and (d) it is the other way round.

Moving B to (from) an odd level number means effectively creating (deleting) that level. Surprisingly, this changes the number of crossings even in subgraphs not connected to B, as can be seen in Fig. 2. Hence, a local update is not possible and we use the efficient bilayer crossing counting algorithm by Barth et al. [3] to count the crossings between these levels. In Fig. 2 the vertex block B is moved to level $l = 5$. Hence, to update the number of crossings we subtract the crossings

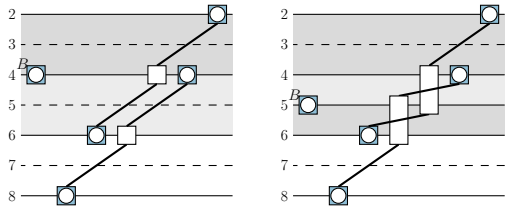


Fig. 2. Modifying $\phi(B)$ changes the crossing count

Algorithm 3: VERTICAL-SWAP$(\mathcal{G}, \mathcal{E}, B, l)$

Input: Block graph $\mathcal{G} = (\mathcal{B}, \mathcal{F})$, current embedding $\mathcal{E} = (\phi, \pi)$, vertex block B to sift on level l
Output: Updated embedding $\mathcal{E} = (\phi, \pi)$, change in crossing number Δ

1 $\Delta \leftarrow 0$
2 **if** l is odd **then** $\Delta \leftarrow \Delta - \texttt{CROSSINGS}(l-3, l-1) - \texttt{CROSSINGS}(l-1, l+1)$
3 **else** $\Delta \leftarrow \Delta - \texttt{CROSSINGS}(l-2, l-1) - \texttt{CROSSINGS}(l-1, l) - \texttt{CROSSINGS}(l, l+2)$
4 $\phi(B) \leftarrow l$
5 update π by placing each edge block that becomes active at the barycenter of its adjacent vertex blocks and bucket-sorting all blocks
6 **if** l is odd **then**
7 $\quad \Delta \leftarrow \Delta + \texttt{CROSSINGS}(l-3, l-1) + \texttt{CROSSINGS}(l-1, l) + \texttt{CROSSINGS}(l, l+1)$
8 **else** $\Delta \leftarrow \Delta + \texttt{CROSSINGS}(l-2, l) + \texttt{CROSSINGS}(l, l+2)$
9 **foreach** active edge block $A \in N^-(B) \cup N^+(B)$ **do**
10 $\quad (\mathcal{E}, \delta) \leftarrow \texttt{HORIZONTAL-STEP}(\mathcal{G}, \mathcal{E}, A)$; $\Delta \leftarrow \Delta + \delta$
11 $(\mathcal{E}, \delta) \leftarrow \texttt{HORIZONTAL-STEP}(\mathcal{G}, \mathcal{E}, B)$; $\Delta \leftarrow \Delta + \delta$
12 **return** $(\mathcal{E}, \Delta)$

between levels 2 and 4 as well as 4 and 6 before the swap (line 2) and add the crossings between 2 and 4, 4 and 5, as well as 5 and 6 (line 7). Note that some of the numbers of crossings after one vertical swap can be reused in the next one.

In contrast, the vertical jump does not count these changes, as all crossings are relative to the current level $l_{\min}$. Hence, only the lines 1, 4, 5 and 9–12 of Algorithm 3 are executed. One problem created by the vertical jump is that the edge blocks of B might have completely new positions when B arrives at its old level. This impedes optimizing the position of B near its old position. We avoid this by starting the vertical step at the current level twice swapping upwards and downwards. We omit this detail in pseudocode just for simplicity.

3.3 Initialization of a Horizontal Sifting Step

As the level assignment ϕ does not change during a horizontal sifting step, the algorithm basically equals the sifting step introduced in global sifting [1].

To improve the performance of one horizontal sifting step [4], it is necessary to keep the adjacency lists $N^-(B, \overline{\phi}(B))$ and $N^+(B, \underline{\phi}(B))$ of each block $B \in \mathcal{B}$

Algorithm 4: HORIZONTAL-STEP($\mathcal{G}, \mathcal{E}, B$)

Input: Block graph $\mathcal{G} = (\mathcal{B}, \mathcal{F})$, embedding $\mathcal{E} = (\phi, \pi)$, block B to sift
Output: Updated ordering π, change in crossing number Δ

1 $p \leftarrow \pi(B); \mathcal{E}_{\text{best}} \leftarrow \mathcal{E}; \Delta_{\text{best}} \leftarrow \infty; \Delta \leftarrow 0; \Delta_{\text{old}} \leftarrow 0$
2 place B at first position of $\mathcal{B}$ and sort the adjacencies of all blocks
3 **for** $i \leftarrow 1$ **to** $|\mathcal{B}| - 1$ **do**
4 **if** $i = p$ **then** $\Delta_{\text{old}} \leftarrow \Delta$
5 $(\mathcal{E}, \delta) \leftarrow$ `HORIZONTAL-SWAP` $(\mathcal{G}, \mathcal{E}, B, \pi^{-1}(i+1))$; $\Delta \leftarrow \Delta + \delta$
6 **if** $\Delta < \Delta_{\text{best}}$ **then** $\mathcal{E}_{\text{best}} \leftarrow \mathcal{E}; \Delta_{\text{best}} \leftarrow \Delta$
7 **return** $(\mathcal{E}_{\text{best}}, \Delta_{\text{best}} - \Delta_{\text{old}})$

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 $I^-(B)$ and $I^+(B)$ of lengths $|I^-(B)| := |N^-(B, \overline{\phi}(B))|$ and $|I^+(B)| := |N^+(B, \underline{\phi}(B))|$, respectively. $I^-(B)$ stores the indices where B is stored in each adjacent block A's adjacency $N^+(A, \underline{\phi}(A))$. More precisely, let $A = N^-(B, \overline{\phi}(B))[i]$ be a $\overline{\phi}(B)$-neighbor of B in direction $-$. Then $I^-(B)[i]$ holds the index at which B is stored in $N^+(A, \underline{\phi}(A))$. Symmetrically, $I^+(B)$ stores the indices at which B is stored in the adjacency $N^-(A, \overline{\phi}(A))$ of each adjacent block A. See Fig. 1(c) for an example. The creation of the four arrays for each block (second part of line 2 of Algorithm 4) can be done in $\mathcal{O}(|E|)$ time by traversing $\mathcal{B}$ once. See Algorithm 6 in Appendix C or [1] for details.

3.4 Horizontal Sifting Step

In a horizontal sifting step (Algorithm 4) all positions i in $\mathcal{B}$ are tested for a (vertex or edge) block $B \in \mathcal{B}$ (lines 3–6) and the best embedding $\mathcal{E}_{\text{best}}$ is returned. Similarly to the vertical step, we only compute the change in the number of crossings when swapping A iteratively with its right neighbor (line 5). To be able to return the change in the number of crossings (line 7) we store the relative number of crossings that B causes at its old position p (line 4).

3.5 Horizontal Sifting Swap

The horizontal 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. Lemma 1 states which segments are involved and Proposition 1 states how the number of crossings changes on such a level. Both can be found in Appendix B and in [1].

Algorithm 5 shows the details of a horizontal sifting swap. First, the levels at which (significant) swaps occur and the direction of the segments changing their crossings are found (lines 4–8). For each entry (l, d) of the set $\mathcal{L}$ the l-neighbors of A and B in direction d are retrieved. Using the notion of G' only the consecutive (dummy) vertices in A and B on level l are swapped. The permutation of the neighboring level $\text{next}^d(l)$ remains unchanged. Thus, the computation of the

Algorithm 5: HORIZONTAL-SWAP$(\mathcal{G}, \mathcal{E}, A, B)$

Input: Block graph $\mathcal{G} = (\mathcal{B}, \mathcal{F})$, embedding $\mathcal{E} = (\phi, \pi)$, consecutive blocks A, B
Output: Updated embedding $\mathcal{E}$, change in crossing count

```
1  begin
2      π' ← π; π'(A) ← π(B); π'(B) ← π(A); E ← (φ, π')
3      if B ∈ B_E ∧ B is not active then return (E, 0)
4      L ← ∅; Δ ← 0
5      if φ̄(A) ∈ levels(B) then L ← L ∪ {(φ̄(A), −)}
6      if φ(A) ∈ levels(B) then L ← L ∪ {(φ(A), +)}
7      if φ̄(B) ∈ levels(A) then L ← L ∪ {(φ̄(B), −)}
8      if φ(B) ∈ levels(A) then L ← L ∪ {(φ(B), +)}
9      foreach (l, d) ∈ L do
10         Δ ← Δ + uswap(A, B, l, d)
11         UPDATE-ADJACENCIES(A, B, l, d)
12     return (E, Δ)
13 function uswap(A, B, l, d): integer
14     let X_0 ≺ ··· ≺ X_{r−1} ∈ N^d(A, l) be the l-neighbors of A in direction d
15     let Y_0 ≺ ··· ≺ Y_{s−1} ∈ N^d(B, l) be the l-neighbors of B in direction d
16     c ← 0; i ← 0; j ← 0
17     while i < r and j < s do
18         if π(X_i) < π(Y_j) then c ← c + (s − j); i ← i + 1
19         else if π(X_i) > π(Y_j) then c ← c − (r − i); j ← j + 1
20         else c ← c + (s − j) − (r − i); i ← i + 1; j ← j + 1
21     return c
```

change in the number of crossings among segments between l and $\text{next}^d(l)$ can be done as in [4], which we adapt to our notation (lines 13–21): The l-neighbors are traversed from left to right. If an l-neighbor of A is found (line 18) the corresponding segment will cross all remaining $s-j$ incident/inner segments of B after the swap. If an l-neighbor of B is found (line 19) the segment has crossed all remaining $r-i$ incident/inner segments of A before the swap. Common neighbors present both cases at the same time (line 20). An update of the adjacency after a swap (line 11) is only necessary if A and B have common l-neighbors. Algorithm 7 in Appendix C and [1] shows how this can be done in overall $\mathcal{O}(\deg(A)+\deg(B))$ time similarly to the function uswap.

3.6 Time Complexity

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

Proof. A horizontal sifting step of a block B needs $\mathcal{O}(|E| \cdot \deg(B))$ time [1]. Hence, (horizontally) sifting an edge block takes $\mathcal{O}(|E|)$ time. A vertical swap

of vertex block B consists of counting the crossings between 5 levels ($\mathcal{O}(|E| \cdot \log|E|)$, [3]), (de-)activating edge blocks ($\mathcal{O}(|E|)$), a horizontal sifting step for each incident edge block ($\mathcal{O}(|E| \cdot \deg(B))$ in total), and the horizontal sifting step of B ($\mathcal{O}(|E| \cdot \deg(B))$). We fix the level radius, i. e., the number of tested levels in each direction, to a constant. Thus, we obtain the time complexity $\mathcal{O}(|E| \cdot \deg(B) + |E| \cdot \log|E|)$ for a vertical step. A sifting round consists of a vertical step of each vertex block and has time complexity $\mathcal{O}(\sum_{B \in \mathcal{B}_V}(|E| \cdot \deg(B) + |E| \cdot \log|E|)) = \mathcal{O}(|E|^2 + |E| \cdot |V| \cdot \log|V|)$, since $\mathcal{O}(\log|E|) = \mathcal{O}(\log|V|)$. □

The time complexity is $\mathcal{O}(|E|^2)$ for dense graphs, i. e., $|E| \geq |V| \log|V|$. Our experiments show that the counting of the crossings can be neglected in practice. The time complexity raises to $\mathcal{O}(|E|^2 \cdot |V| + |E| \cdot |V|^2 \cdot \log|V|)$ in total using an unfixed level radius.

4 Vertical Compaction

Similarly to [7], we apply a postprocessing step to reduce the number of levels without changing the crossing number. In the level embedding we search for distinct (non-necessarily monotonous) cuts from the left to the right which consist solely of dummy vertices and do not cross outer segments. We delete these cuts, i. e., its vertices, and lift the subgraph below each cut by one level. Even our naive implementation needs less than 5% of the overall running time.

5 Experimental Results

For each $|V| \in \{25, 50, \ldots, 400\}$ we generated 10 random graphs with $|V|$ vertices and an edge set drawn from all $4|V|$-element subsets of the edge set of the complete graph. We use a random injective initial leveling $\phi : V \to \{1, \ldots, |V|\}$ giving the algorithms some freedom to place vertices. We compared the algorithms iterative one-sided 2-level barycenter (BC) [13], global sifting (GlS) [1], upward planarization layout (UPL) [7], and our grid sifting algorithm with the level radii 3 (GrS3), 10 (GrS10) and unconfined (GrS*). GS21 uses radius 21 but new levels only, such that it holds the invariant of having only one vertex per level and tests the same number of levels as GrS10.

We execute 8 rounds for each grid sifting variant since then there is no further significant improvement in the number of crossings. We apply UPL 20 times on each input to choose the best result. Clearly, BC and GlS are the fastest algorithms by far, but they cannot change the leveling. To ensure that their results do not suffer from saving running time, we execute (actually unreasonable) 400 sweeps and sifting rounds. All benchmarks ran on an Intel Core 2.8 GHz machine, where UPL was a binary C executable and the other algorithms were implemented in Java within Gravisto [2].

As UPL is slower than our grid sifting algorithms (Fig. 3), we could test UPL only for graphs with up to 275 vertices. All grid sifting variants give less crossings

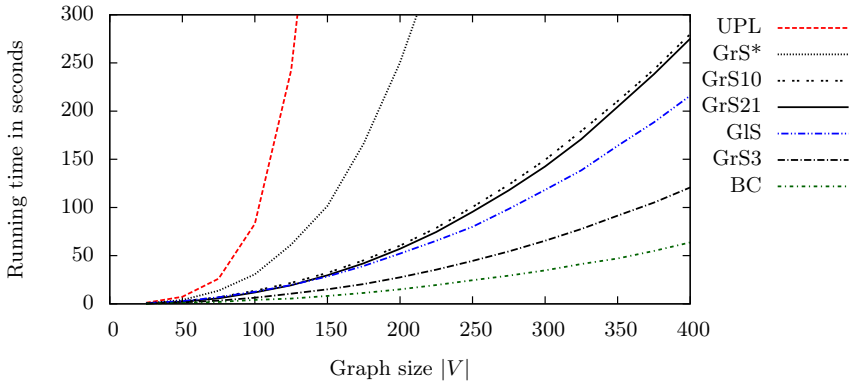


Fig. 3. Running times of the algorithms

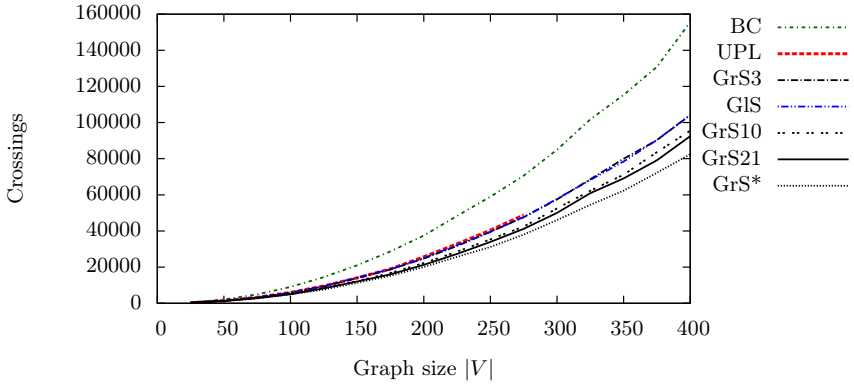


Fig. 4. Number of crossings of the algorithms

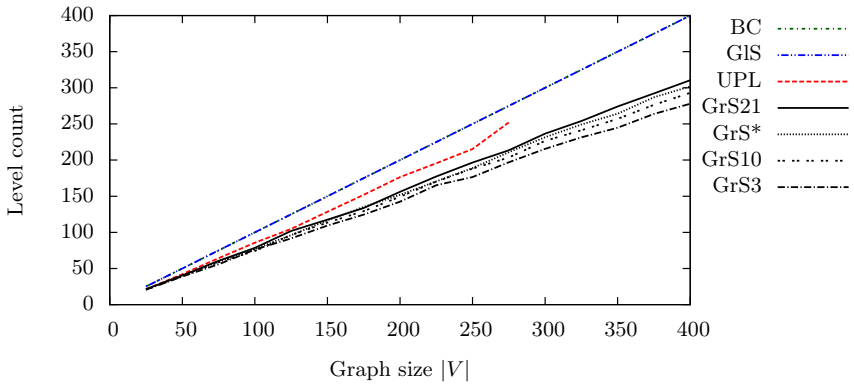


Fig. 5. Number of levels of the algorithms

(Fig. 4) and levels (Fig. 5) than UPL. GrS21 is a good tradeoff between running time and result. The Wilcoxon signed rank test [17] indicates with a confidence of 99% that GrS21 yields 7.9% less crossings than UPL. The number of levels is rather high, as we are optimizing the number of crossings first.

6 Summary

Combining the leveling and crossing reduction phases of the Sugiyama framework delivers few crossings. The planarization approach by Chimani et al. [6,7] and the presented two-dimensional grid sifting follow this idea. The latter delivers similar results regarding levels and crossings to the former, however, its time complexity is roughly $\mathcal{O}(|E|^2)$ instead of $\mathcal{O}(|E|^5)$. We suggest to use planarization for sparse (see Appendix D) and grid sifting for more dense graphs. Our implementation is available under the GPL in Gravisto [2].

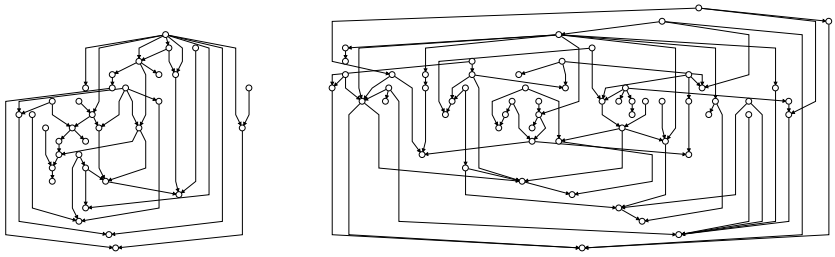
References

1. 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, *WALCOM 2010*, volume 5942 of *LNCS*, pages 70–81. Springer, 2010.
2. C. Bachmaier, F. J. Brandenburg, M. Forster, P. Holleis, and M. Raitner. Gravisto: Graph visualization toolkit. In J. Pach, editor, *GD 2004*, volume 3383 of *LNCS*, pages 502–503. Springer, 2004. `http://gravisto.fim.uni-passau.de/`.
3. W. Barth, P. Mutzel, and M. Jünger. Simple and efficient bilayer cross counting. *J. Graph Alg. App.*, 8(2):179–194, 2004.
4. M. Baur and U. Brandes. Crossing reduction in circular layout. In J. Hromkovic, M. Nagl, and B. Westfechtel, editors, *WG 2004*, volume 3353 of *LNCS*, pages 332–343. Springer, 2004.
5. U. Brandes and B. Köpf. Fast and simple horizontal coordinate assignment. In P. Mutzel, M. Jünger, and S. Leipert, editors, *GD 2001*, volume 2265 of *LNCS*, pages 31–44. Springer, 2002.
6. 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.
7. M. Chimani, C. Gutwenger, P. Mutzel, and H.-M. Wong. Upward planarization layout. In D. Eppstein and E. R. Gansner, editors, *GD 2009*, volume 5849 of *LNCS*, pages 94–106. Springer, 2010.
8. E. G. Coffman Jr. and R. L. Graham. Optimal scheduling for two processor systems. *Acta Informatica*, 1(3):200–213, 1972.
9. G. Di Battista, A. Garg, G. Liotta, 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. graphs available at `http://www.graphdrawing.org/`.
10. P. Eades and N. C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11(1):379–403, 1994.
11. E. R. Gansner, E. Koutsofios, S. North, and K.-P. Vo. A technique for drawing directed graphs. *IEEE Trans. Software Eng.*, 19(3):214–230, 1993.
12. M. Jünger, E. K. Lee, P. Mutzel, and T. Odenthal. A polyhedral approach to the multi-layer crossing minimization problem. In G. Di Battista, editor, *GD 1997*, volume 1353 of *LNCS*, pages 13–24. Springer, 1997.

13. M. Kaufmann and D. Wagner. *Drawing Graphs*, volume 2025 of *LNCS*. Springer, 2001.
14. C. Matuszewski, R. Schönfeld, and P. Molitor. Using sifting for k-layer straightline crossing minimization. In J. Kratochvíl, editor, *GD 1999*, volume 1731 of *LNCS*, pages 217–224. Springer, 1999.
15. 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.
16. 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.
17. F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

A Example Drawings

Exemplarily, we visualize the embeddings of two Rome graphs [9] computed by grid sifting. The edge directions are determined as described in Appendix D. For the fourth phase of the framework, i. e., for computing real coordinates, we applied the algorithm of Brandes and Köpf [5].



(a) grafo2379.35 with 35 vertices and 45 edges on 17 levels with 6 crossings

(b) grafo4562.60 with 60 vertices and 79 edges on 19 levels with 27 crossings

Fig. 6. Drawings of two Rome graphs

B Details of Horizontal Swaps

In this section we use the perception of G' with dummy vertices and segments again.

Lemma 1. *Let $\mathcal{B}$ be the block list in the current order. Let $B \in \mathcal{B}$ be the successor of $A \in \mathcal{B}$. 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 type 2 conflicts are absent, at least one of the segments of a crossing is an outer segment. Let $(u, v), (w, z) \in E'$ be two segments between the same levels with $u \neq w$ and $v \neq z$. If the two segments cross after swapping A and B, but did not cross before (or vice versa) either u and w or v and z 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 v and z were swapped and thus u and w were not, $\phi(v) = \phi(z)$ 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 u and w nor v and z would have been swapped. The other case of swapping u and w instead of v and z is symmetric. □

Proposition 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* $\text{block}(u)$ *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.*

C Algorithms

With Algorithm 6 we build up the sorted adjacency of each block before each horizontal step. We traverse the blocks B in the current order of $\mathcal{B}$ and add B to the next free position j of the cleared adjacency array $N^+(A, \underline{\phi}(A))$ ($N^-(C, \overline{\phi}(C))$) of each incoming $\overline{\phi}(B)$-neighbor A (outgoing $\underline{\phi}(B)$-neighbor C). Both values for $I^+(A)$ and $I^-(B)$ ($I^+(B)$ and $I^-(C)$) and their positions are only known after the second traversal of a segment e. Thus, we cache the first array position j as an attribute p of e. Let $v \in V'$ be a dummy vertex of an edge block B with $\phi(v) = l$. We explicitly do not update any outgoing l-neighbor adjacency of v if $\overline{\phi}(B) \leq l < \underline{\phi}(B)$ and no incoming adjacency of v if $\overline{\phi}(B) < l \leq \underline{\phi}(B)$. These vertices and arrays are only implicit for performance reasons. They only would contain one element $N^{\cdot}(B, l)[0] = B$. Thus, associated values "$I^{\cdot}(B, l)$" are not needed in Algorithm 7. For B only two arrays $I^-(B)$ and $I^+(B)$ remain representing the position of B in its incident vertex blocks.

After a horizontal swap of two blocks A and B we adjust the adjacency arrays of common neighbors with Algorithm 7.

D Additional Benchmarks

Analogously to Chimani et al. [7], we additionally compare the algorithms on the widely used Rome graphs library [9] in Figs. 7 to 9. 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 from former to later vertex definitions, see [7]. The remaining parameter setup is identical to Sect. 5.

As the Rome graphs are very sparse and almost planar, the planarization approach UPL results in fewer crossings than grid sifting. However, grid sifting produces fewer levels. The running times are comparable.

Algorithm 6: SORT-ADJACENCIES($\mathcal{G}, \pi$)

Input: Block graph $\mathcal{G} = (\mathcal{B}, \mathcal{F})$, ordering π
Output: Ordered sets $N^-(B, \overline{\phi}(B))$, $N^+(B, \underline{\phi}(B))$, $I^{\cdot}(B)$ for each block $B \in \mathcal{B}$

for $i \leftarrow 0$ **to** $|\mathcal{B}| - 1$ **do**
  $\pi(\mathcal{B}[i]) \leftarrow i$; clear arrays $N^-(\mathcal{B}[i], \overline{\phi}(\mathcal{B}[i]))$, $N^+(\mathcal{B}[i], \underline{\phi}(\mathcal{B}[i]))$ and $I^{\cdot}(\mathcal{B}[i])$
foreach vertex block or active edge block $B \in \mathcal{B}$ **do**
  foreach $e = (A, B) \in \mathcal{F}$ **do**
    if A is an inactive edge block **then** $A \leftarrow A'$ with unique $(A', A) \in \mathcal{F}$
    add B to the next free position j of $N^+(A, \underline{\phi}(A))$
    if $\pi(B) < \pi(A)$ **then** $p[e] \leftarrow j$ *// first traversal of e*
    else $I^+(A)[j] \leftarrow p[e]$; $I^-(B)[p[e]] \leftarrow j$ *// second traversal of e*
  foreach $e = (B, C) \in \mathcal{F}$ **do**
    if C is an inactive edge block **then** $C \leftarrow C'$ with unique $(C, C') \in \mathcal{F}$
    add B to the next free position j of $N^-(C, \overline{\phi}(C))$
    if $\pi(B) < \pi(C)$ **then** $p[e] \leftarrow j$ *// first traversal of e*
    else $I^+(B)[p[e]] \leftarrow j$; $I^-(C)[j] \leftarrow p[e]$ *// second traversal of e*

Algorithm 7: UPDATE-ADJACENCIES(A, B, l, d)

Input: Consecutive blocks $A, B \in \mathcal{B}$, level l, direction d
  $N^d(A, l), I^d(A), N^d(B, l), I^d(B)$
Output: Updated adjacencies of A and B and all common neighbors

let $X_0 \prec \cdots \prec X_{r-1} \in N^d(A, l)$ be the l-neighbors of A in direction d
let $Y_0 \prec \cdots \prec Y_{s-1} \in N^d(B, l)$ be the l-neighbors of B in direction d
$i \leftarrow 0; j \leftarrow 0$
while $i < r$ and $j < s$ **do**
  if $\pi(X_i) < \pi(Y_j)$ **then** $i \leftarrow i + 1$
  else if $\pi(X_i) > \pi(Y_j)$ **then** $j \leftarrow j + 1$
  else
    $Z \leftarrow X_i$ *// $= Y_j$*
    swap entries at pos. $I^d(A)[i]$ and $I^d(B)[j]$ in $N^{-d}(Z, l)$ and in $I^{-d}(Z)$
    $I^d(A)[i] \leftarrow I^d(A)[i] + 1$; $I^d(B)[j] \leftarrow I^d(B)[j] - 1$
    $i \leftarrow i + 1; j \leftarrow j + 1$

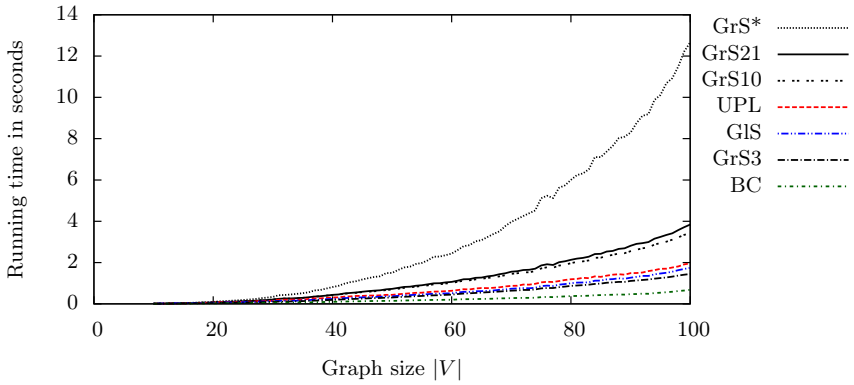


Fig. 7. Running times of the algorithms

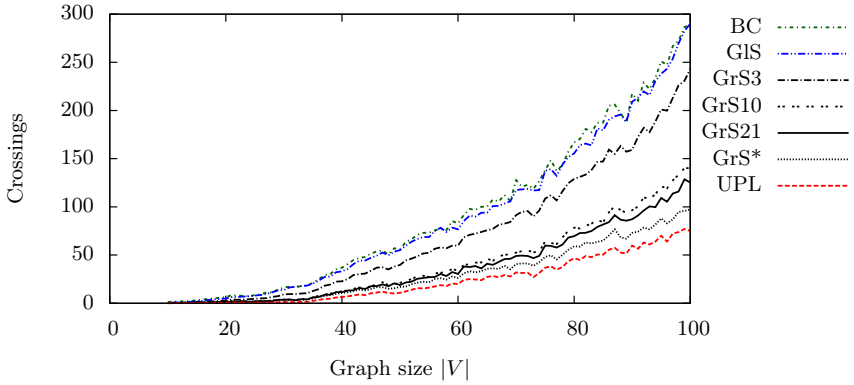


Fig. 8. Number of crossings of the algorithms

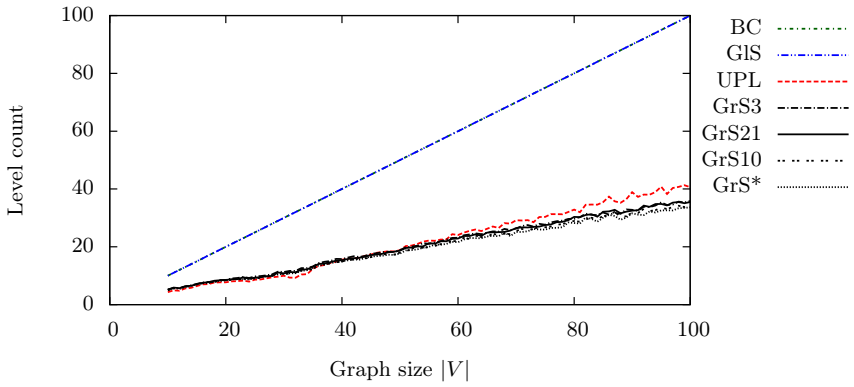


Fig. 9. Number of levels of the algorithms