

# Crossing Reduction for Hierarchical Graphs with Intra-Level Edges

Technical Report MIP-0608

July, 2006

Christian Bachmaier and Michael Forster

University of Passau, Germany

94030 Passau, Germany

Fax: +49 851 509 3032

{bachmaier,forster}@fmi.uni-passau.de

**Abstract.** In drawings of hierarchical graphs generated by the conventional Sugiyama framework the vertices are positioned on multiple horizontal level lines. This drawing style which allows edges only between vertices on different levels is well suited for the visualization of a common direction of flow from lower to higher levels in a graph.

In this paper we are interested in reordering the vertices on each level line to increase readability of the drawing, i. e., in reducing the number of edge crossings. As novelty, we additionally allow the existence of edges with both end vertices on a common level, which often occur in practice. Experimentally we found out, that we can save about 30% of the crossings compared to the existing standard heuristic which ignores those edges.

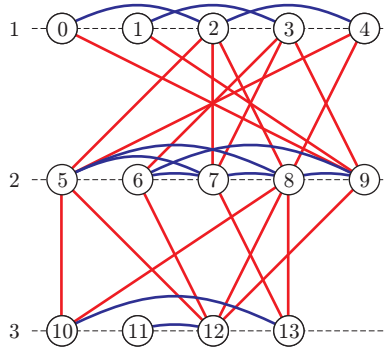
## 1 Introduction

In hierarchical graph layout vertices are usually drawn on parallel horizontal lines, and edges are drawn as strict  $y$ -monotone polylines that may bend when they intersect a level line. The standard drawing framework [12] consists of four phases: *cycle removal* (reverses appropriate edges to eliminate cycles), *level assignment* (assigns vertices to levels such that no edge has both end vertices on the same level and introduces dummy vertices to represent edge bends), *crossing reduction* (permutes vertices on the levels), and *coordinate assignment* (assigns  $x$ -coordinates to vertices,  $y$ -coordinates are implicit through the levels). See [7] for an extended overview.

In this paper we are especially interested in the crossing reduction phase to improve readability [10] of the drawing. As novelty we allow the existence of *horizontal* edges having both end vertices on a common level and take their crossings into account. This type of level graphs often occurs in practice: for example graphs where the level assignment is already predefined, e. g., by breadth first search (BFS) levels to express distances, or social network graphs [2–4]

where the importance (*centrality*) of an *actor* (modeled by a vertex) defines its level.

Preferably we want to draw all edges as straight lines. But at least with vertices having more than two adjacent horizontal edges this is not possible, unless we allowed overlapping of edges and crossings between vertices and edges. To avoid this, cf. Fig. 1, we draw horizontal edges as semicircles with different radii. Further we restrict ourselves to draw the semicircles only above the level lines and not below, what practically makes it easier to attack the problem. Since this leads to three kinds of crossings, our idea is to treat them separately with a standard sifting heuristic, with circular crossing reduction, and a new counting algorithm. Since all three algorithms are some kind of sifting, we interlock them all in a new integrated sifting heuristic.



**Fig. 1.** Level graph example illustrating drawing conventions

This paper is organized as follows: After some necessary preliminaries showing existing algorithms in Sect. 2 we explain in Sect. 3 how to extend sifting also to count crossings generated by horizontal edges.

## 2 Preliminaries

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

## 2.1 Sifting with a Crossing Matrix

The most common technique for crossing reduction in level graphs is to only consider two consecutive levels at a time in multiple top-down and bottom-up passes. Starting with an arbitrary ordering of the first level, subsequently the ordering of one level is fixed, while the next one is reordered to minimize the number of crossings in-between. This *one-sided two-level crossing reduction problem* is  $\mathcal{NP}$ -hard [6] and well-studied: Given a 2-level graph  $G = (V_1 \dot{\cup} V_2, E, \phi)$  where  $V_1$  is the set with the fixed ordering, the objective is to compute an ordering of the second level  $V_2$  which causes fewest crossings. As not mentioned otherwise, we restrict ourselves to one-sided two-level crossing reduction in the following.

To face the problem, we use the *sifting* heuristic here, which is slower than simple heuristics like *barycenter* or *median* heuristics [7] but generates less crossings. Barycenter (median) heuristic assigns each vertex of  $V_2$  the barycenter (median) value of its neighbors in  $V_1$ , assuming the positions of vertices in  $V_1$  are numbered from 1 to  $|V_1|$  according to  $\prec$ . A sorting according to this values defines the ordering among the vertices in  $V_2$ . Sifting was originally introduced as a heuristic for vertex minimization in ordered binary decision diagrams [11] and later adapted for the one-sided crossing minimization problem [9]. The idea is to keep track of the objective function while moving in a *sifting step* a vertex  $u \in V_2$  along a fixed ordering of all other vertices in  $V_2$  and then placing  $u$  to its locally optimal position. The method is thus an extension of the greedy-switch heuristic [5], where  $u$  is swapped iteratively with its successor. We call a single swap a *sifting swap*. Executing a sifting step for every vertex in  $V_2$  is called a *sifting round*. For crossing reduction the objective function is the number of crossings between the edges incident to the vertex under consideration and all other edges. The efficient computation of the crossing count in sifting is based on the *crossing matrix*. Its  $|V_2|^2$  entries correspond to the number of crossings caused by (the edges of) pairs of vertices in a particular relative ordering and can be computed as a preprocessing step in  $\mathcal{O}(|E|^2)$  [13,14]. Whenever a vertex is placed to a new position, only a smallish number of updates is necessary which allows a running time of  $\mathcal{O}(|V_2|^2)$  for one round. Usually only few sifting rounds (3 – 5 for reasonable problem instances) are necessary to reach a local optimum for all vertices simultaneously. Thereby the largest reduction of crossings usually occurs in the first round.

## 2.2 Circular Sifting

However, the asymptotic overall running time of the algorithm described above is  $\mathcal{O}(|E|^2 + |V_2|^2)$  and too high for our purposes. Thus we apply the heuristic of Baur and Brandes [1] originally used for the  $\mathcal{NP}$ -hard [8] circular crossing reduction problem: Reorder the vertices  $V$  of a graph  $G = (V, E)$  which all are placed on a single circle (e. g., in Fig. 3) to minimize the number of crossings among the straight-line edges in  $E$ . Since there is no “circular” ordering, Baur and Brandes define linear orders  $\prec_\alpha$  by selecting a reference vertex  $\alpha \in V$  which is the first of

the (here counter-clockwise) sequence. For finding the locally optimal position for a vertex  $u \in V$  in a sifting step it is sufficient to record the change in crossing count while swapping  $u$  with its successor  $v$ . This can be done by considering only edges incident to  $u$  or  $v$ : After a swap exactly those pairs of these edges cross which did not before. All other crossings remain unchanged (let  $\chi(\pi)$  be the number of crossings of a drawing  $\pi$  and  $N(u)$  be the set of adjacent vertices of  $u \in V$ ):

**Lemma 1 (Baur, Brandes).** *Let  $u \prec_u v_p \in V$  be consecutive vertices in a circular layout  $\pi$  and let  $\pi'$  be the layout with their positions swapped, then*

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

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

We adapt the above idea to one-sided two-level crossing reduction, which we call *vertical sifting* for simplicity. We mainly exchange  $\prec_\alpha$  by  $\prec$  as illustrated in Algorithms 1, 2, and 3 and thus consider the ordering of the neighbors on the fixed level 1. We obtain the same results as with the matrix method, however, without knowing the absolute crossing numbers. Since all three methods are generic and also fit for the following algorithms, Algorithm 2 already contains lines 4 and 8. But up to now these lines can be ignored and the input graphs should be considered as  $G = (V_1 \dot{\cup} V_2, E, \phi)$  for ease of understanding. For efficiency reasons all shown operations should be implemented in place on the graph data structure.

---

#### Algorithm 1: SIFTING-ROUND

---

**Input:** Ordered two level graph  $G = (V_1 \dot{\cup} V_2, E, H, \phi)$

**Output:** Updated ordering of  $V_2$

```

1 foreach  $u \in V_2$  do
2    $\lfloor V_2 \leftarrow \text{SIFTING-STEP}(G, u)$ 
3 return  $V_2$ 

```

---

### 3 Reduction with Horizontal Edges

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

---

**Algorithm 2: SIFTING-STEP**

---

**Input:** Ordered two level graph  $G = (V_1 \dot{\cup} V_2, E, H, \phi)$ , Vertex  $u \in V_2$  to sift  
**Output:** Updated ordering of  $V_2$

- 1 let  $v_0 = u \prec v_1 \prec \dots \prec v_{|V_2|-1}$  denote the current ordering of  $V_2$
- 2 **foreach**  $v \in V_2$  **do**
- 3     Sort  $E(v) \subseteq E$  by ascending ordering of  $V_1$  in  $\mathcal{O}(|E|)$  time
- 4     Sort  $H_l(v), H_r(v) \subseteq H$  by ascending ordering of  $V_2$  in  $\mathcal{O}(|H|)$  time
- 5  $\chi \leftarrow 0; \chi^* \leftarrow 0$  *// current and best number of crossings*
- 6  $p^* \leftarrow 0$  *// best vertex position*
- 7 **for**  $p \leftarrow 1$  **to**  $|V_2| - 1$  **do**
- 8      $l \leftarrow \text{UPDATE-HORIZ-ADJ}(G, u, v_p)$
- 9      $\chi \leftarrow \chi + \text{SIFTING-SWAP-V}(G, u, v_p)$
- 10     **if**  $\chi < \chi^*$  **then**
- 11          $\chi^* = \chi$
- 12          $p^* = p$
- 13 **return**  $V_2 \leftarrow v_1 \prec \dots \prec v_{p^*-1} \prec u \prec v_{p^*} \prec \dots \prec v_{|V_2|-1}$

---



---

**Algorithm 3: SIFTING-SWAP-V**

---

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

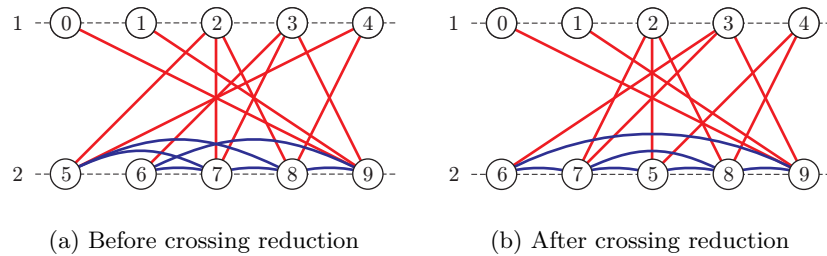
- 1 let  $x_0 \prec \dots \prec x_{r-1}$  be the neighbors of  $u$  in  $V_1$ ,  $\{x_i, u\} \in E$
- 2 let  $y_0 \prec \dots \prec y_{s-1}$  be the neighbors of  $v_p$  in  $V_1$ ,  $\{y_j, v_p\} \in E$
- 3  $c \leftarrow 0; i \leftarrow 0; j \leftarrow 0$
- 4 **while**  $i < r$  **and**  $j < s$  **do**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

---

The one-sided two-level crossing reduction problem on an extended 2-level graph is  $\mathcal{NP}$ -hard too, since at least two subproblems, considering only the vertical edges [6] and considering only the horizontal edges [8] are  $\mathcal{NP}$ -hard as well. As a consequence, we use extensions of the sifting heuristic for an efficient solution of the problem. To provide a better overview, we first outline the reduction of different crossing types separately.

### 3.1 Crossings Between Horizontal Edges

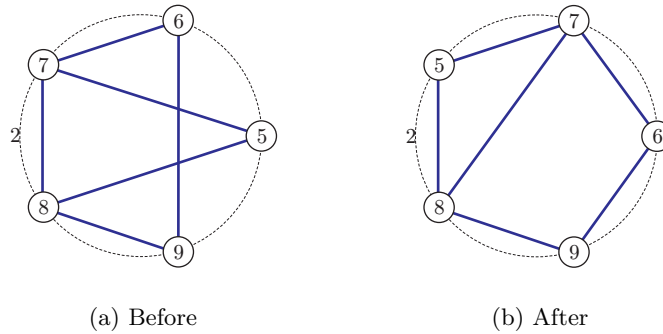
We consider overlapping horizontal edges  $\{v_1, v_4\}, \{v_2, v_3\} \in H$  with  $v_1 \prec v_2 \prec v_3 \prec v_4$  (also if  $v_1 = v_2$  and/or  $v_3 = v_4$ ) as not crossing, since we draw them as circular arcs without any crossing (except common end points) instead of straight horizontal lines: For an edge  $\{u, v\} \in H$  we use a quadratic spline with an amplitude, i. e., the height of the only control point, raising with the number of vertices between  $u$  and  $v$  in the given ordering  $\prec$  of  $V_2$ . Thereby we take care not to introduce unnecessary “double” crossings between horizontal and vertical edges by restricting the maximum edge amplitude according to the dimension of the drawing. For example, if the edge  $\{5, 8\}$  in Fig. 2(a) would have a higher amplitude, it would cross the edge  $\{5, 4\}$ . Note that we require to draw all horizontal edges completely above the second level line, as will be explained later in Sect. 3.2.



**Fig. 2.** The first 2 levels of Fig. 1

A straightforward solution to the horizontal crossing reduction problem is the introduction of dummy levels which contain an artificial dummy vertex splitting each horizontal edge. Then, in worst case, the amplitudes of the edges may be pairwise different, what leads up to  $|H|$  different dummy levels. As a consequence – to be able to run later a 2-level crossing reduction algorithm which considers all type of crossings – each vertical edge must be split in  $|H| + 1$  segments by  $|H|$  new dummy vertices. This prevents not only time efficient processing, but also is obstructive to get a good result, i. e., less crossings, since each of the additionally necessary  $|H|$  crossing reduction rounds is a heuristic only and thus not exact.

Thus for *horizontal sifting* we again decided to take the circular sifting algorithm of [1] already used for crossing reduction among vertical edges in Sect. 2.2. Considering the horizontal line of level 2 bent to a circle (see Fig. 3), the circular crossing reduction algorithm fits “out of the box”: For one round call Algorithm 1 with line 9 of Algorithm 2 updated to call Algorithm 4 instead of Algorithm 3. Line 3 of Algorithm 2 can be left away in this case. Principally, Algorithm 4 is the same as Algorithm 3 except that the neighbors are on level 2 and the ordering  $\prec$  is replaced by  $\prec_{v_p}$ , i. e., the ordering of  $V_2$  is different in each swap!



**Fig. 3.** Circular crossing reduction for horizontal edges for the graph in Fig. 2

With Algorithm 5 we keep the ordered horizontal adjacencies of vertex  $u$  up to date during a sifting step. Thus we know the ordering  $\prec_{v_p}$  among  $u$ 's neighbors, since this is nothing else than the concatenation of  $H_r(u)$  and  $H_l(u)$  (in this order). Therefore we need no reordering for determining the  $x_i$ s per swap. The same holds for the  $y_i$ s: Algorithm 5 also updates the horizontal adjacencies of vertex  $v_p$ , but does in contrast to  $u$  not maintain their ordering due to performance restrictions. But we rely on the fact, that an edge  $h = \{u, v_p\}$  is always the first of  $H_l(v_p)$ . This is true, since we build up the sorting of this adjacency list right after  $u$  was put on the first position of  $V_2$  in Algorithm 2 and then there never was an update to this ordering. In other words, the ordering of the horizontal adjacencies of all vertices  $v_p$  is valid throughout the complete sifting step besides obsolete positions of edges  $\{u, v_p\}$ . However, this exceptions are irrelevant for the determination of the orderings of the  $y_i$ s, since they never contain  $u$ .

### 3.2 Crossings Between Vertical and Horizontal Edges

We restrict horizontal edges only to be routed above the second level line. Otherwise, if we allowed routing on both sides simultaneously, the number of crossings between vertical and horizontal edges would depend on the vertical edges

---

**Algorithm 4: SIFTING-SWAP-H**

---

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

```

1 let  $x_0 \prec_{v_p} \dots \prec_{v_p} x_{r-1}$  be the neighbors of  $u$  in  $V_2 - \{v_p\}$ ,  $\{x_i, u\} \in H$ 
2 let  $y_0 \prec_{v_p} \dots \prec_{v_p} y_{s-1}$  be the neighbors of  $v_p$  in  $V_2 - \{u\}$ ,  $\{y_j, v_p\} \in H$ 
3  $c \leftarrow 0$ ;  $i \leftarrow 0$ ;  $j \leftarrow 0$ 
4 while  $i < r$  and  $j < s$  do
5   if  $x_i \prec_{v_p} y_j$  then
6      $c \leftarrow c - (s - j)$ 
7      $i \leftarrow i + 1$ 
8   else if  $y_j \prec_{v_p} x_i$  then
9      $c \leftarrow c + (r - i)$ 
10     $j \leftarrow j + 1$ 
11  else
12     $c \leftarrow c - (s - j) + (r - i)$ 
13     $i \leftarrow i + 1$ ;  $j \leftarrow j + 1$ 
14 return  $c$ 

```

---



---

**Algorithm 5: UPDATE-HORIZ-ADJ**

---

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

```

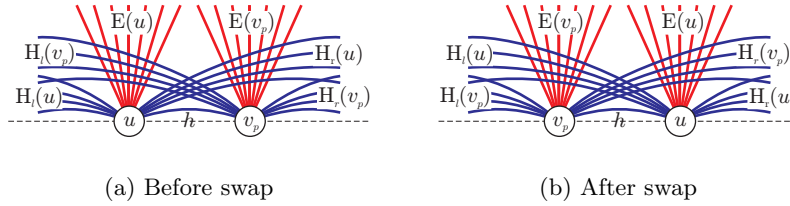
1  $l \leftarrow 0$  // number of short horizontal edges
2 while  $\{u, v_p\} = \text{getFirst}(H_r(u)) \in H$  do
3    $h \leftarrow \text{removeFirst}(H_r(u))$ 
4    $\text{append}(H_l(u), h)$ 
5    $\text{removeFirst}(H_l(v_p))$  // first, since never updated before
6    $\text{prepend}(H_r(v_p),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

```

---

to vertices on a third level, which contradicts the pairwise level by level sweep approach.

Like in the above algorithms, swapping vertex  $u$  with its successor  $v_p$  changes only crossings (here among vertical and horizontal edges) between edges incident to  $u$  or  $v_p$ . Thus for computing the change in the crossing count we only need the sizes of the six sets  $H_l(v)$ ,  $H_r(v)$ ,  $E(v)$  with  $v \in \{u, v_p\}$ , cf. Fig. 4.



**Fig. 4.** Crossings among horizontal and vertical edges

Neglecting potentially existing edges  $h = \{u, v_p\} \in H$  which are a non contributing special case, we obtain (1) as change in crossing count when swapping  $u$  and  $v_p$ . The correctness follows again from the invariant that after a swap exactly those pairs of horizontal (except  $h$ ) and vertical edges cross which did not before.

$$(|H_r(v_p)| - |H_l(v_p)|) \cdot |E(u)| + (|H_l(u)| - |H_r(u)|) \cdot |E(v_p)| \quad (1)$$

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

---

**Algorithm 6:** SIFTING-SWAP-HV

---

**Input:** Ordered two level graph  $G = (V_1 \dot{\cup} V_2, E, H, \phi)$ , Swap vertices  $u, v_p \in V_2$ , Number of short edges  $l = |\{\{u, v_p\} \in H\}|$

**Output:** Change in crossing count

**1 return**  $((|H_r(v_p)| - l) - |H_l(v_p)|) \cdot |E(u)| +$   
**2**  $((|H_l(u)| - l) - |H_r(u)|) \cdot |E(v_p)|$

---

### 3.3 All Crossings

The modular design of the above algorithms now pay off: For *extended sifting* we call Algorithm 1 with an updated line 9 of Algorithm 2 in order to call Algorithm 7, where we simply add the three independent changes in crossing counts. However, other formulas preferring some type of crossings at the expense of more crossings of other types are possible, e. g., the usage of weighting factors.

---

#### Algorithm 7: SIFTING-SWAP-EXT

---

**Input:** Ordered two level graph  $G = (V_1 \dot{\cup} V_2, E, H, \phi)$ , Swap vertices  $u, v_p \in V_2$

**Output:** Change in crossing count

- 1  $c_1 \leftarrow \text{SIFTING-SWAP-V}(G, u, v_p)$
  - 2  $c_2 \leftarrow \text{SIFTING-SWAP-H}(G, u, v_p)$
  - 3  $c_3 \leftarrow \text{SIFTING-SWAP-HV}(G, u, v_p)$
  - 4 **return**  $c_1 + c_2 + c_3$
- 

Theorem 1 shows, that we obtain the same time bound as sifting for a graph  $G = (V, E, \phi)$  considering only vertical edges or for a graph  $G = (V, H)$  considering only horizontal edges.

**Theorem 1.** *One round of extended one-sided sifting on an extended 2-level graph  $G = (V, E, H, \phi)$  needs  $\mathcal{O}(|V| \cdot (|E| + |H|))$  time.*

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

One round of vertical resp. horizontal sifting needs  $\mathcal{O}(|V||E|)$  resp.  $\mathcal{O}(|V||H|)$  time according to Theorem 3 of [1]. One round of mixed sifting needs  $\mathcal{O}(|V||H|)$  time, since one step needs  $\mathcal{O}(|H|)$  time: The initial sorting of the horizontal adjacency in Algorithm 2 can be done in  $\mathcal{O}(|H|)$  by traversing the vertices of  $V_2$  in order and adding each to the adjacency list of its right/left neighbors. One of the  $|V_2|$  sifting swaps needs constant time.

The interlocked execution is possible, since the only updates to the horizontal adjacency list are done by Algorithm 2, thus the algorithms mutually don't bias.  $\square$

### 3.4 Sweep over all Levels

According to our experience, the quality of sifting does not depend much on the quality of the initial vertex ordering. However, a "bad" initialization raises the number of needed sifting rounds and thus the absolute running time. Thus it maybe useful to apply at least some rounds of horizontal sifting to  $V_1$  to get a practical initial ordering.

In the top-down sweep we reorder the levels  $i$  from 2 to  $k$  by consecutively applying our extended one-sided two-level crossings reduction on the fix ordered set  $V_{i-1}$  and on the permutable set  $V_i$ . In the subsequent bottom-up sweep we reorder the levels  $i$  from  $k-1$  down to 1 by consecutively applying the two-level algorithm on the fix ordered set  $V_{i+1}$  and the permutable set  $V_i$ . But bottom-up we have a different situation, since the horizontal edges are below the current level  $i$  and cross edges from level  $i$  and  $i-1$ , see Fig. 5 for an example. However, the formula for crossings of horizontal and vertical edges (1) does not depend on any vertex ordering different to that on level  $i$  and especially does not depend on that of level  $i-1$ . Thus we count the change in the number of crossings of the horizontal edges of level  $i$  with the vertical edges between level  $i$  and  $i-1$  during a swap. For this we let for every  $v \in V_i$  be  $E(v) = \{(x, v) \mid x \in V_{i-1}\}$  instead of  $\{(y, v) \mid y \in V_{i+1}\}$  in Algorithm 6, which then does nothing else than it does when executed during a top-down sweep. After some iterations, say 10, of top-down with subsequent bottom-up sweeps the algorithm terminates.

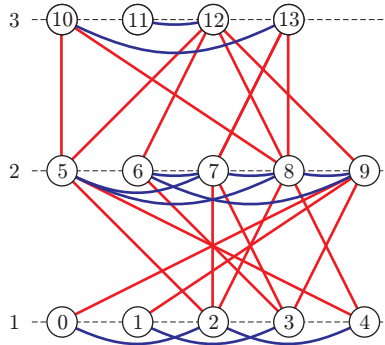


Fig. 5. Bottom-up sweep for the graph in Fig. 1

### 4 Experimental Results

To analyse the performance of one sifting round of our one-sided two-level crossing reduction heuristics, we have implemented them in Java. Further, we have realized the corresponding standard sifting algorithm which uses a crossing matrix to compare its practical running time with the sifting algorithm of [1]. We have tested the implementations using a total number of 13750 random graphs: 25 graphs for each combination of the parameters  $|V_1| = |V_2| \in \{50, 100, 150, \dots, 1100\}$ ,  $|E|/|V_2| \in \{1, \dots, 5\}$ , and  $|H|/|V_2| \in \{1, \dots, 5\}$ .

Figure 7 and 8 confirms that it makes sense to consider all types of crossings simultaneously, since the algorithm generates (as expected) fewer crossings than standard sifting, experimentally by a factor of 0.7. This is a very encouraging result, since the differences in absolute running times between our extending

sifting and the existing standard vertical sifting and horizontal sifting, i. e., the running time of mixed sifting, are negligible in practice even on larger graphs, see Fig. 6. Just to give a feeling, e. g., the running time of extended sifting on a graph with  $|V_1| = |V_2| = |E| = |H| = 10^4$  is about 13 minutes.

## 5 Conclusion

We extended the well known sifting heuristic for crossing reduction to extended level graphs. Ignoring self loops which do not affect the crossing number, our algorithm works out of the box also on multi graphs within the same time complexity.

As far we only use a random initial ordering of the vertices, but the quality of the ordering produced by extended sifting is not independent of the quality of the initial orderings of all levels. Thus it may be helpful to use some extensions of fast and simple heuristics, e. g., barycenter or median [7] heuristics, to reduce the crossing count of the input.

Future research on this topic can be heuristics, which use the freedom of routing horizontal edges above and below their level line and not restricting them to one side.

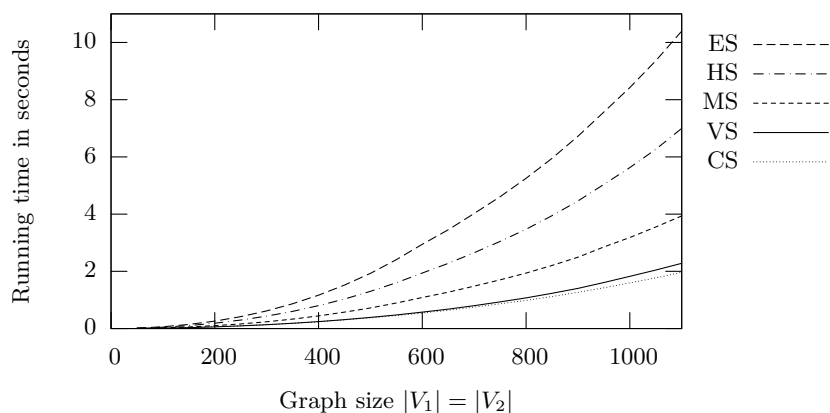
## References

- [1] 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, 2005.
- [2] U. Brandes and T. Erlebach, editors. *Network Analysis, Methodological Foundations*, volume 3418 of *LNCS Tutorial*. Springer, 2005.
- [3] U. Brandes, P. Kenis, and D. Wagner. Centrality in policy network drawings. In J. Kratochvíl, editor, *Proc. Graph Drawing, GD 1999*, volume 1731 of *LNCS*, pages 250–258. Springer, 1999.
- [4] U. Brandes, P. Kenis, and D. Wagner. Communicating centrality in policy network drawings. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):241–253, 2003.
- [5] P. Eades and D. Kelly. Heuristics for reducing crossings in 2-layered networks. *Ars Combinatorica*, 21(A):89–98, 1986.
- [6] P. Eades and N. C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11(1):379–403, 1994.
- [7] M. Kaufmann and D. Wagner. *Drawing Graphs*, volume 2025 of *LNCS*. Springer, 2001.
- [8] S. Masuda, T. Kashiwabara, K. Nakajima, and T. Fujisawa. On the  $\mathcal{NP}$ -completeness of a computer network layout problem. In *Proc. IEEE International Symposium on Circuits and Systems*, pages 292–295, 1987.
- [9] 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.

- [10] H. C. Purchase. Which aesthetic has the greatest effect on human understanding? In G. Di Battista, editor, *Proc. Graph Drawing, GD 1997*, volume 1353 of *LNCS*, pages 248–261. Springer, 1997.
- [11] 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.
- [12] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981.
- [13] V. Valls, R. Marti, and P. Lino. A branch and bound algorithm for minimizing the number of crossing arcs in bipartite graphs. *Journal of Operational Research*, 90:303–319, 1996.
- [14] A. Yamaguchi and A. Sugimoto. An approximation algorithm for the two-layered graph drawing problem. In T. Asano, H. Imai, T. Lee, S. Nakano, and T. Tokuyama, editors, *Proc. International Conference on Computing and Combinatorics, COCOON 1999*, volume 1627 of *LNCS*, pages 81–91. Springer, 1999.

## A Benchmark Results

The following figures provide benchmark results comparing the heuristics to reduce crossings: vertical sifting (CS with crossing matrix, VS without), horizontal sifting (HS), mixed sifting (MS), and extended sifting (ES). All benchmarks were run on a 2.6 GHz Pentium PC under the Java 2 platform 5.0 from Sun Microsystems, Inc.



**Fig. 6.** Benchmark: running times

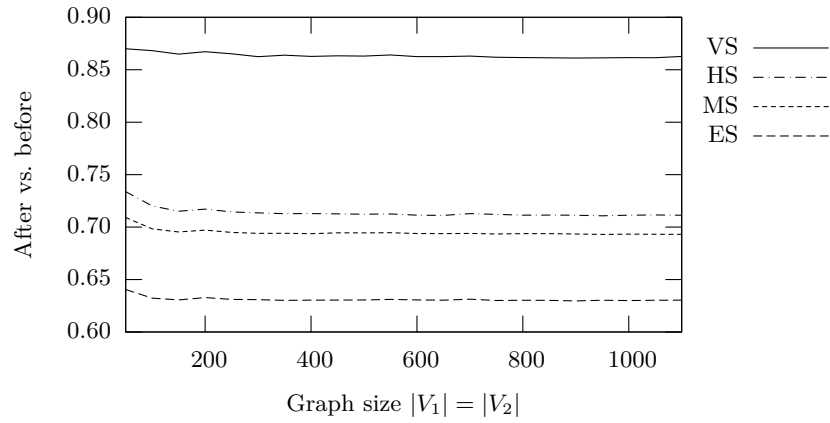


Fig. 7. Benchmark: total crossing numbers

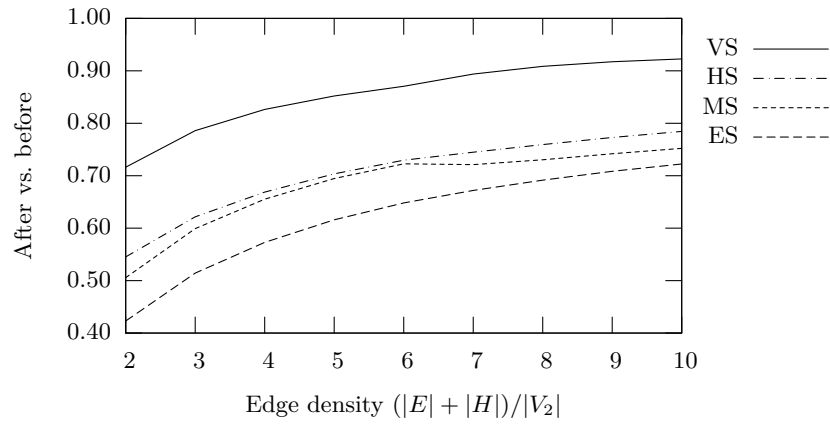


Fig. 8. Benchmark: total crossing numbers

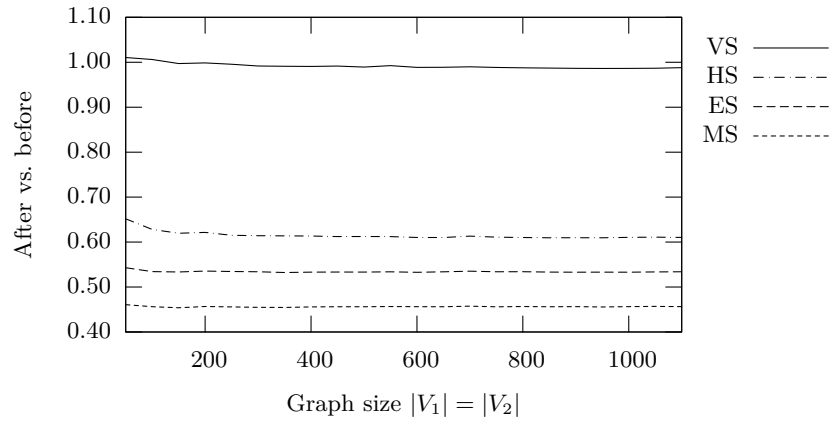


Fig. 9. Benchmark: numbers of crossings between vertical and horizontal edges

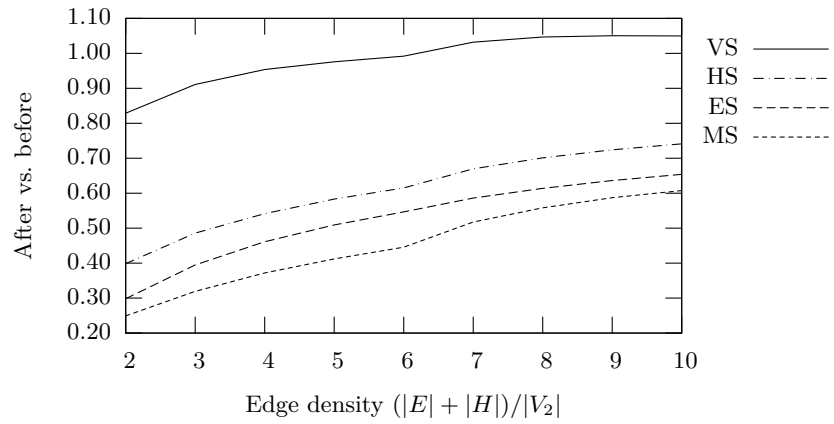
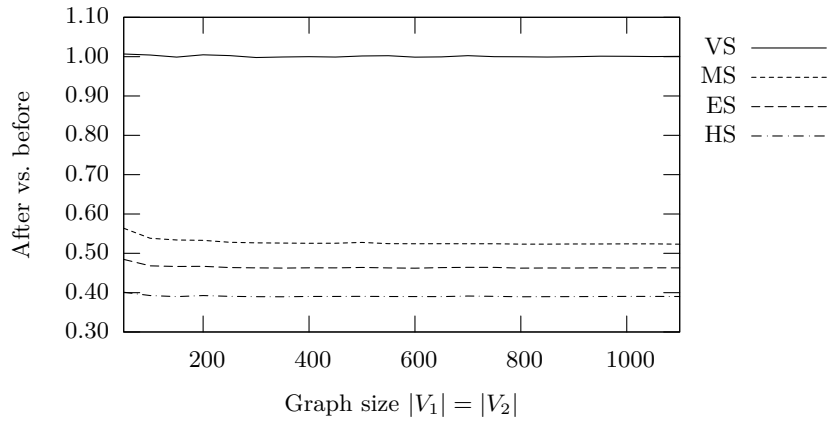
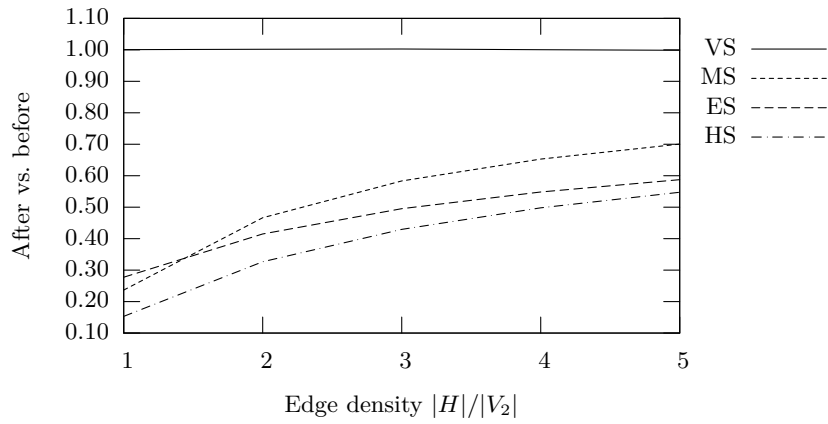


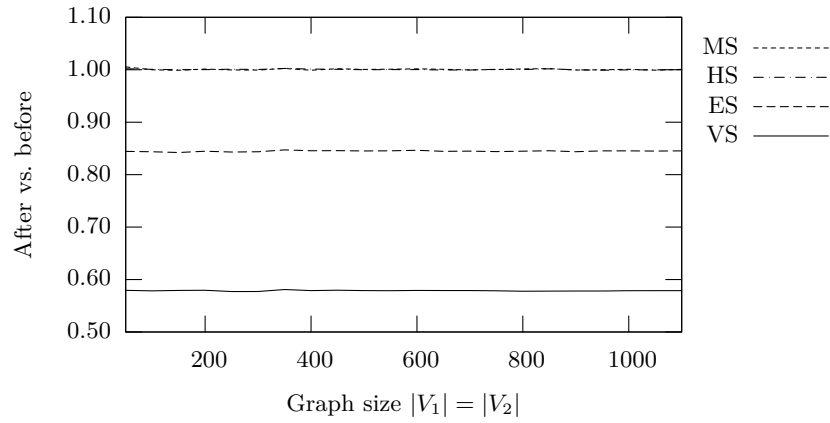
Fig. 10. Benchmark: numbers of crossings between vertical and horizontal edges



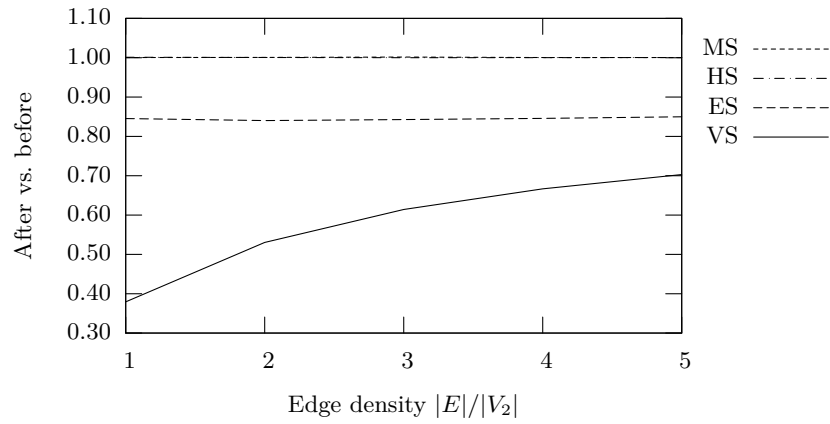
**Fig. 11.** Benchmark: numbers of crossings between horizontal edges



**Fig. 12.** Benchmark: numbers of crossings between horizontal edges



**Fig. 13.** Benchmark: numbers of crossings between vertical edges



**Fig. 14.** Benchmark: numbers of crossings between vertical edges