Bottleneck Crossing Minimization in Layered Graphs

Matthias F. Stallmann and Saurabh Gupta

Computer Science Department, NC State University, Raleigh, NC 27695 USA
`matt_stallmann@ncsu.edu` `saurabh.gupta.7@gmail.com`,

Abstract. Extensive research over the last twenty or more years has been devoted to the problem of minimizing the *total* number of crossings in layered directed acyclic graphs. Algorithms for this problem are used for graph drawing, to implement one of the stages in the multi-stage approach proposed by Sugiyama et al. In some applications, such as minimizing the deleterious effects of crosstalk in VLSI circuits or minimizing the total number of crossings in subgraphs, it may be more appropriate to minimize the *maximum* number of crossings involving any edge. We refer to this as the *bottleneck* crossing problem. In this paper we propose a new heuristic designed specifically for the bottleneck problem and describe experimental results that demonstrate its superiority over the barycenter method on various classes of multi-layer graphs. In some cases, our new heuristic performs better than the barycenter heuristic even when the total number of crossings is the measure of merit.
Software (C source) is available for the heuristics discussed here and others.

1 Background

An ℓ-layer graph $G = (V, E)$ has $V = V_1 \oplus ... \oplus V_\ell$ and $E \subseteq \bigcup_{1 \leq i < \ell}(V_i \times V_{i+1})$. In other words, the nodes are partitioned into ℓ *layers* and all edges connect vertices on adjacent layers. It is usually assumed that the graph is directed and acyclic and a directed edge edge vw has $v \in V_i$ and $w \in V_{i+1}$ for some i. We use the notation $E(v)$ to denote the edges incident on v.

An *embedding* of a layered graph G assigns the nodes of V_i to points on the line $y = i$ and a permutation π_i to V_i so that, for each $v \in V_i$, the node v is mapped to the point $(\pi_i(v), i)$. We call $\pi_i(v)$ the *position* of v (on its layer) and use the simpler notation $p(v)$ from here on. If each $vw \in E$ is mapped to a straight line (or arrow), an embedding induces crossings among the edges. In particular, the edge vw *crosses* xy – assuming $v, x \in V_i$, $w, y \in V_{i+1}$ – if $p(v) < p(x)$ and $p(w) > p(y)$. The *crossing number* of *edge* vw is the number of edges xy that cross vw – denote this as $c(vw)$. The well-known crossing number problem is one of minimizing the total number of crossings, or $\sum_{e \in E} c(e)/2$. This is motivated, among other contexts, by the desire to create esthetically pleasing drawings of graphs [1, Chapter 9] and minimizing crosstalk in VLSI circuits [2].

Our focus, however, is on the *bottleneck crossing problem*, which is to minimize $\max_{e \in E} c(e)$. We refer to this as simply the *bottleneck* problem from here on and the original crossing number problem as the *total* problem. For convenience in later discussion, we use the notation $c(E')$ to mean $\max_{e \in E'} c(e)$, where $E' \subseteq E$.

It is easy to show that the bottleneck problem, like the total problem is NP-hard. The proof by Garey and Johnson in [3] for the total problem can be adapted to use the Bandwidth problem [4, Appendix A1.3] instead of Minimum Linear Arrangement.

There are two key applications of the bottleneck problem, both involving large graphs. One is the VLSI crosstalk problem as described in [2] – the real problem is a bottleneck problem rather than a total problem: maximum delay and probability of a defect in a circuit are both related to maximum crosstalk along a single wire. See also [5] for a more detailed description of how wire crossings degrade the performance of a circuit. The other application is *activity-based management (ABM)*: here the nodes represent products and services and an edge vw means that v is an input to w, i.e., w makes use of v in some fashion. The usual graph drawing techniques apply when a manager wants to get a sense of the flow of the products and services. Sometimes, however, a smooth zoom into a subgraph is required. Minimizing $c(E)$ ensures that all subgraphs will have relatively few crossings.

2 The Maximum Crossings Edge Heuristic

To minimize the maximum number of crossings of any edge, it is natural to focus on edges that have the maximum number of crossings and attempt to reduce their number of crossings. This requires a dynamic approach wherein the algorithm keeps track of $c(e)$ for all edges e throughout. Every *iteration* uses sifting as described in [6, 7] on an endpoint v of the edge vw with maximum $c(vw)$. In contrast to regular sifting we want a position for v that minimizes $c(vw)$ while also keeping $c(e)$ for edges e incident on $layer(v)$ small.

A *pass* of our heuristic, the *maximum crossings edge (mce)* heuristic,[1] sifts every node exactly once, as described in Fig 1. An important point in the sifting process, see Fig 2, is the interpretation of best position. The presumed best position of x for edges incident on $layer(x)$ is updated locally, considering only the edges incident on the two nodes being swapped. So the EdgeSift algorithm may not find the real position of minimum bottleneck. Fig 3 illustrates this. Suppose edge $1, 6$ is chosen and node 6 on the upper level is being sifted. When we swap node 6 with 5, the number of crossings for edge $1, 5$ increases by 2 to become 6, i.e., the value of c_5 becomes 6. Later, when 6 is swapped with 3, the value of c_3 is 3, because that is the maximum number of crossings *among the*

[1] The heuristic is inspired by and takes its name from a similar dynamic sifting algorithm for minimizing total crossings, the *maximum crossings node* heuristic, reported by Gupta [8].

edges incident to 3 and 6 and the resulting best position $p = 3$, even though it induces 6 bottleneck crossings, not the true minimum.

One might consider the failure to find the true minimum position a drawback of the heuristic. It turns out, however, to be an advantage. The true best position in Fig 3 is the starting position, a common phenomenon when there are many crossings to start with. Insisting on a true bottleneck minimum for the current sift is therefore likely to trap the heuristic in a local optimum. It is desirable to promote movement of nodes, especially in the early stages. Our implementation of sifting takes this one step further: we choose a position farthest away from the starting position when breaking ties.

```
while there is an edge with at least one unmarked endpoint do
    let $e = vw$ be an edge with $c(e) = c(E)$
    if $v$ is not marked then EdgeSift($v$)
    if $w$ is not marked then EdgeSift($w$)
    mark both $v$ and $w$
end do
```

Fig. 1. A pass of the maximum crossings edge algorithm.

```
EdgeSift($x$)  is
    let $y_1, \ldots, y_k$ be the nodes on $layer(x)$ sorted by position
    maintain $c =$ the best number of bottleneck crossings so far
        and $p =$ the position at which $c$ occurred
    for $i = p(x) - 1$ downto $1$ do
        swap $x$ with $y_i$ and update $c(e)$ for $e \in E(x) \cup E(y_i)$
        let $c_i = c(E(x) \cup E(y_i))$
        if $c_i < c$ then let $c = c_i$ and $p = i$
    end do
    repeat the preceding sift loop for $i = 1$ to $k$
    if $p < p(x)$ then move $x$ before $y_p$
    else if $p > p(x)$ then move $x$ after $y_p$
```

Fig. 2. Sifting a node to minimize the crossings of its edges.

We turn now to the problem of updating $c(e)$ for $e \in E(x) \cup E(y_i)$ when x and y_i are swapped. To simplify the discussion, consider a generic swap between two nodes v and w in adjacent positions on the same layer: let $p(v) < p(w)$ before the swap and $p(v) > p(w)$ after it. The algorithm is essentially the same as the $O(|E| + |C|)$ inversion-counting algorithm of Barth et al. [9]. However, instead

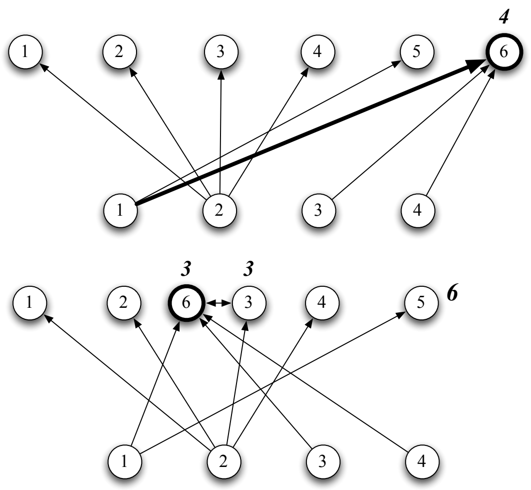

Fig. 3. The minimum position for a node is not optimal.

of merely counting inversions among the relevant edges, we need to update $c(e)$ for each one.

Let us restrict our attention to the subset of $E(v) \cup E(w)$ incident on the layer above that of v, w. The same procedure is applied (separately) for the edges incident on the layer below. To reflect the fact that v will no longer be to the left of w, we decrement $c(e)$ and $c(f)$ if e crosses f when $p(v) < p(w)$. Conversely, we increment $c(e)$ and $c(f)$ if e crosses f when $p(v) > p(w)$. The swapping algorithm (for one of the two neighboring layers) is described in Fig 4.

```
SWAP($v, w$) **is**
    let $v_1, \ldots, v_{deg(v)}$ be the nodes adjacent to $v$, sorted by position
    let $w_1, \ldots, w_{deg(w)}$ be the nodes adjacent to $v$, sorted by position
    do an insertion sort by position,
        starting with the order $v_1, \ldots, v_{deg(v)}, w_1, \ldots, w_{deg(w)}$:
    **whenever** there is an inversion between $v_i$ and $w_j$ **do**
        decrement $c(vv_i)$ and $c(ww_j)$

    repeat the sort, starting with $w_1, \ldots, w_{deg(v)}, v_1, \ldots, v_{deg(v)}$,
        and incrementing instead of decrementing the $c$'s
```

Fig. 4. Updating edge crossing counts when swapping two nodes.

If we maintain a priority queue of the edges, the time bound for choosing an edge with maximum $c(e)$ is $O(|E| \log |E|)$ per pass. The remaining dominant component of the time bound is the total number of update (increment/decrement) operations during swapping. A sifting operation with node x will swap x with each node on $layer(x)$ at most twice. The total number of updates per iteration is therefore some constant times $deg(x) \sum_{y \in layer(x)} deg(y)$. For a pass, this translates to $\sum_{1 \leq i \leq \ell} \sum_{x,y \in E_\ell} deg(x)deg(y)$ or $|E|^2$. Thus the total time for a pass is $O(|E|^2)$. Based on [9] the total time for a barycenter sweep is $|E| \log |V|$ – assuming we need to update the total number of crossings at the end of every iteration to determine the minimum so far. For a fair comparison we need to count the time per iteration amortized over a pass/sweep. This gives us $O(d|E|)$ for mce versus $O((|E| \log |V|)/\ell)$ for barycenter, where d is the average degree of a node. In almost all situations this analysis favors the barycenter heuristic, since there are likely to be more than $\log |V|$ layers.

3 Experiments

What follows is a description of the experiments we use to demonstrate the effectiveness of the mce heuristic on the bottleneck problem, and, in some cases, the total crossings problem.

3.1 Problem Instances

The instances we use are of three kinds. First there are instances that have exactly 100 nodes from the set of Rome graphs [10], a total of 140 of them. Layering for these graphs was accomplished using the heuristic of Gansner et al. [11], as extracted from the outputs of experiments conducted by Chimani et al. [12][2] Dummy nodes were added to ensure true layered graphs. Table 1 gives the important characteristics of our versions of the Rome graphs.

	nodes	edges	density	max deg	layers	nodes/layer min	nodes/layer max
mean	144.79	180.01	1.24	7.81	8.09	1.69	43.84
stdev	16.91	21.31	0.05	1.12	2.19	1.49	7.69
min	115	144	1.13	6	4	1	21
median	140	176	1.25	8	7	1	44
max	192	244	1.41	11	14	11	69

Table 1. Statistics for 100-node Rome graphs.

Next there are random trees that are generalizations of those proposed by Stallmann et al. in [13]. The basic idea is to compute a minimum spanning tree on a random set of points in the unit square and partition it into k layers. With

[2] Obviously we are not using the layer-free drawings of that paper but the Sugiyama drawings used for comparison of crossing numbers.

only two layers this is easy: just assign each point a random layer and make sure the tree alternates between them (you can also control the number of points in each layer). With $\ell > 2$ layers we take a different approach. We compute the tree and assign layers as we go, ensuring that the layer sequence for each path is a subsequence of $1, 2, \ldots, \ell, \ell-1, \ldots, 1, 2, \ldots$. Nodes are reasonably distributed over the layers but there is no guarantee on how many will appear in each layer. A tree class is denoted by $t_{n,\ell}$, where n is the number of nodes and ℓ the number of layers.

Finally, there are random layered dags designed to approximate those of the ABM application (as used in [14]). We choose $\ell =$ number of layers, $k =$ number of nodes per layer, and $P =$ the probability that a specific edge will be included. There are $(\ell-1)k^2$ potential edges, so the expected number of edges is $P(\ell-1)k^2$. To make these dags more interesting and realistic we ensure that every node has at least one predecessor. This adds $(\ell-1)k(1-P)^k$ to the expected number of edges – an edge is added if none of the k potential predecessors of a node is chosen in the usual way. The number of nodes per layer also varies a lot in ABM, but we chose not to deal with this additional complication. Nomenclature for these random layered dags is $d_{\ell,k,d}$, where ℓ is the number of layers, k the number of nodes per layer, and d the density, i.e., number of edges divided by number of nodes.

Aside from the 140 Rome graphs, all other graph classes in our experiments consisted of 100 independent random instances. Our approach for tree and random dag classes was to focus on ones that have the same density as the Rome graphs (except for trees) and the same ratio of number of layers to nodes per layer. We chose graphs with four times as many nodes as the Rome graphs: the basic tree class was $t_{560,14}$ and the basic dag class was $d_{14,40,1.25}$ – statistics for these are shown in Table 2 and Table 3, respectively.

						nodes/layer	
	nodes	edges	density	max deg	layers	min	max
mean	560	559	1.0	4.0	14	18.8	51.5
stdev	0	0	0	0	0	2.8	3.2
min	560	559	1.0	4.0	14	9	46
median	560	559	1.0	4.0	14	19	51
max	560	559	1.0	4.0	14	25	63

Table 2. Statistics for the class $t_{560,14}$.

We investigated the dependence of the results on several variables: (a) instance size with the same layers to nodes/layer ratio ($d_{7,20,1.25}$, $d_{28,80,1.25}$); (b) different ratios, inverting the ratio as in $d_{40,14,1.25}$, $t_{560,40}$; (c) different densi-

	nodes	edges	density	max deg	layers	nodes/layer min	nodes/layer max
mean	550.0	679.8	1.23	7.9	14	40	40
stdev	2.2	13.4	0.03	0.9	0	0	0
min	544	650	1.18	7	14	40	40
median	550	678	1.23	8	14	40	40
max	554	731	1.34	11	14	40	40

Table 3. Statistics for the class $d_{14,40,1.25}$. The number of nodes is less than the expected 560 because isolated nodes on the first layer have been deleted.

ties[3] ($d_{14,40,1.04}$, $d_{14,40,1.5}$, $d_{14,40,2.0}$); and (d) different variances in node degree[4], achieved by controlling the choice of predecessor for nodes that failed to receive one the usual way. That choice could range from a random one among all potential predecessors to no choice at all. The former yields uniform degree while the latter may introduce a single high-degree node on each layer.

3.2 Experimental Results

The results reported here compare the mce heuristic with the well-known barycenter heuristic [15]. Similar comparisons can be made with other heuristics, including the median heuristic [16], sifting [6, 7] and some new heuristics reported in [8]; preliminary results indicate that, in our context, the barycenter heuristic does at least as well as any of these with respect to the bottleneck problem, both with respect to bottleneck crossings and with respect to total crossings.

We precede each heuristic with a depth-first search starting at an arbitrary node and then sort each layer by the preorder numbers of the nodes. Results for bottleneck and total crossings are significantly better with dfs preprocessing, more so for barycenter than for mce.

Recall that the barycenter heuristic takes a *layer sweep* approach, wherein there is an upward sweep during which π_{i-1} is fixed and π_i altered for $i = 2, \ldots, \ell$ followed by a downward sweep with π_{i+1} fixed and π_i altered for $i = \ell - 1, \ldots, 1$. When π_i is altered, the positions of nodes in V_i are sorted based on the average of the positions of their neighbors in V_{i-1} (upward sweep) or V_{i+1} (downward sweep). These two sweeps constitute a *pass* and each sort is called an *iteration*.

While an mce pass consists of $|V|$ sorting iterations, a barycenter pass consists of only $2\ell - 2$ iterations. Ordinarily, the barycenter heuristic checks for improvement (of the best solution so far) at the end of a pass. The same holds for the default setting of mce. In order to compare the two heuristics fairly, we decided to let each run for the same number of iterations (10,000) and report both the best bottleneck number and best total number over all iterations.

[3] The smallest achievable density while still maintaining connectivity in our random dag model was roughly 1.04.

[4] Here we mean the difference between minimum and maximum degree.

Our basis for comparison for both bottleneck and total crossings is the minimum number achieved by the barycenter heuristics divided by the minimum achieved by mce. The number of bottleneck and total crossings can vary a great deal even within a single class and certainly among different classes, but this ratio remains remarkably stable.

As expected, the mce heuristic consistently outperforms barycenter with respect to bottleneck crossings. Mean ratios range from 1.21 for the dags having highest density to 2.00 for the sparsest dags. The dependence on density should not be a surprise. With dense dags there is not much scope for improvement in number of total crossings – as already observed in [13] – nor, as observed here, for bottleneck crossings. See Table 4 for details.

When the densest and sparsest graphs are ignored, the ratios are consistently around 1.60, both within classes and throughout. In other words, graph size and ratio of layers to nodes per layer have little, if any, effect. The Rome graphs have a slightly higher ratio (1.84), as do the smallest random dags (1.72). Since the latter are the same size as the former, it is natural to conjecture that smaller graphs favor mce. However, the converse is not true: mce does as well on the largest graphs as it does on the mid-range ones. Thus a correlation with graph size is ruled out, at least if we keep aspect ratio the same. This is significant as the dags in both applications – circuit layout and activity-based management – are large and sparse.

	bottleneck crossings		total crossings	
graph class	*bary/mce*	*min(bary,mce)*	*bary/mce*	*min(bary,mce)*
$d_{14,40,1.04}$	2.00 (0.47)	5.7 (1.0)	**0.79** (0.15)	249.5 (62.8)
Rome	1.84 (0.30)	8.3 (2.8)	1.22 (1.17)	265.0 (118.7)
$d_{7,20,1.25}$	1.72 (0.29)	6.1 (1.1)	1.11 (0.76)	186.5 (47.7)
$d_{14,40,1.25}$	1.66 (0.14)	13.5 (1.1)	0.95 (0.08)	1,566.2 (210.9)
$d_{28,80,1.25}$	1.62 (0.11)	35.2 (2.3)	0.87 (0.4)	14,594.3 (812.6)
$d_{40,14,1.25}$	1.57 (0.22)	5.5 (0.6)	0.92 (0.08)	464.6 (57.9)
$t_{560,14}$	1.56 (0.50)	13.5 (1.1)	**0.57** (0.24)	74.6 (19.5)
$t_{560,40}$	1.48 (0.62)	1.9 (0.4)	**0.54** (0.30)	16.8 (9.1)
$d_{14,40,1.5}$	1.47 (0.11)	23.0 (2.0)	0.98 (0.06)	3,715.8 (369.1)
$d_{14,40,2.0}$	1.21 (0.07)	40.1 (3.4)	0.91 (0.03)	10,099.8 (691.0)

Table 4. Results for all of the standard graph classes, sorted by mean value of the ratio of barycenter bottleneck crossings to mce bottleneck crossings. The numbers in parentheses are standard deviations.

Surprisingly, standard deviation among different initial orderings of the *same instance* from a class – different *presentations* [13] – is almost identical to that of different random instances (with random presentations) from the same class. This is especially remarkable in light of the depth-first search preprocessing, which one would expect to smooth over presentation differences.

The behavior of the heuristics with respect to *total number of crossings* exhibits some oddities that may be the basis for conjectures and further investigation. Much of what we observe may demonstrate weaknesses of the barycenter heuristic rather than strengths of mce, but those weaknesses may hold with other heuristics that, unlike mce, are static in the choice of node or layer to use for sorting (see [8] for some dynamic alternatives).

First, it needs to be pointed out that in every class we investigated, including many not reported here, mce outperformed barycenter with respect to total crossings on a significant fraction ($1/4 - 1/3$) of the instances. In contrast, except for the two tree classes, mce outperformed barycenter with respect to bottleneck crossings *on every instance*, usually with a ratio exceeding 1.2.

That being said let us consider the classes for which the performance ratio was worst for mce, the two tree classes and the sparsest dag class. We might conjecture that the cause is a relatively small number of total crossings, but this would fail to explain the good performance of mce on $d_{7,20,1.25}$. Another suggestion is that the barycenter heuristic is expected to do well on trees – subtrees will naturally cluster as the heuristic progresses. The mce heuristic, on the other hand spends a lot of iterations rearranging endpoints of edges with very few crossings.

		bottleneck crossings		total crossings	
choices	max deg	*bary/mce*	*min(bary,mce)*	*bary/mce*	*min(bary,mce)*
40	6.5 (0.8)	2.00 (0.47)	5.7 (1.0)	**0.79** (0.15)	249.5 (62.8)
20	7.5 (0.9)	2.07 (0.39)	6.1 (0.9)	**0.84** (0.17)	264.3 (60.5)
8	10.3 (1.2)	2.16 (0.38)	6.8 (1.0)	**0.90** (0.20)	253.3 (67.9)
4	14.0 (1.4)	2.02 (0.45)	8.0 (1.4)	**0.99** (0.23)	229.2 (69.3)
2	20.9 (1.6)	1.97 (0.46)	8.9 (1.7)	**1.10** (0.40)	265.6 (57.3)
1	32.9 (1.6)	2.39 (2.05)	8.8 (4.3)	**1.16** (0.60)	78.2 (44.0)

Table 5. The effect of maximum degree on relative performance of barycenter versus mce, based on graph classes that are degree-controlled versions of $d_{14,40,1.04}$.

This leads to an interesting conjecture: As the maximum degree in a sparse graph increases, the performance of mce for total crossings should improve. Table 5 bears this out. Maximum degree turns out to be a significant factor, not so much for bottleneck crossings but for total crossings. The first column in the table represents the number of choices for predecessors of the "orphaned" nodes, those that did not obtain predecessors via the usual random choices. The 40 means that all possible nodes on the preceding layer were equally likely to be chosen; a 1 means that all orphaned nodes on a layer would choose the *same* predecessor. When there are a small number of nodes with large degree, barycenter may not be able to distinguish "correctly" between an average position of two nodes that are far apart and an average of the many neighbors of a high-degree node.[5] The mce heuristic, instead, will focus immediately on edges crossing those incident to a high-degree node. This phenomenon needs to be investigated in more detail.

4 Summary

We have presented a new crossing-minimization heuristic whose primary purpose is to minimize the maximum number of crossings induced by any edge, i.e., the bottleneck crossing number. Experimental results demonstrate that the performance of the new maximum crossings edge heuristic is superior to that of barycenter on a large variety of graph classes encompassing several thousand individual graphs. The mce heuristic appears to achieve better results with respect to total crossings on some graphs. The reason for this is not completely clear, but further investigation may lead to better crossing minimization heuristics for both problems and methods for *instance profiling*, i.e., using easily computable (usually static) information about a graph to determine which of several heuristics is likely to achieve the best results.

Software for the heuristics described in this paper, other heuristics, and multiple versions of preprocessing combined with other options, is available from the first author (contact `matt_stallmann@ncsu.edu`). The usage description for the main program looks like

```
Usage: min_crossings [opts] file.dot file.ord
 where opts is one or more of the following
  -h (bary | mod_bary | mcn | sifting | mce) [main heuristic]
  -p (bfs | dfs | mds) [preprocessing - default none]
  -i max_iterations [default heuristic stopping criterion]
  -c iteration [capture the order after this iteration in a file]
  -w (none | avg | left) [adjust weights in barycenter, default avg]
  -s (layer | degree | random) [sifting variation - see paper]
  -e (nodes | edges | early) [mce variation]
  -t if trace printout is desired
```

An *ord* file (with `.ord` extension) gives the order of the nodes on each layer. The input order may be random or come from another heuristic. The program produces an output ord file named using the base name of the input file with suffixes indicating the heuristic(s) used.

Also available are a variety of scripts for generating input files, running multiple experiments, and gathering data.

Acknowledgments The authors would like to thank Hoi-Ming Wong at the University of Dortmund, Germany, for providing GML representations of layouts of the Rome graphs.

[5] The relationship between barycenter performance and maximum degree is considered on a theoretical level by Li and Stallmann [17] and is related.

References

1. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice Hall (1999)
2. Bhatt, S., Leighton, F.: A framework for solving VLSI graph layout problems. JCSS **28** (1984) 300–343
3. Garey, M.R., Johnson, D.S.: Crossing Number is NP-complete. SIAM J. Algebraic Discrete Methods **4** (1983) 312–316
4. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman (1979)
5. Takahashi, H., Keller, K., Le, K., Saluja, K., Takamatsu, Y.: A method for reducing the target fault list of crosstalk faults in synchronous sequential circuits. IEEE Transaction on CAD **24** (2005) 252–263
6. Matuszewski, C., Schönfeld, R., Molitor, P.: Using sifting for k-layer straightline crossing minimization. In: Proc. Graph Drawing 1999. Number 1731 in Lecture Notes in Computer Science (1999) 217–224
7. Schönfeld, R.: k-layer straightline crossing minimization by speeding up sifting. In: Proc. Graph Drawing 2000. (2000)
8. Gupta, S.: Crossing minimization in k-layer graphs. Master's thesis, North Carolina State University (Dec 2008)
9. Barth, W., Jünger, M., Mutzel, P.: Simple and efficient bilayer cross counting. In: Proc. Graph Drawing 2002. Number 2528 in Lecture Notes in Computer Science (2002) 130–141
10. DiBattista, G., Garg, A., Liotta, G., Tammasia, R., Tassinari, E., Vargiu, F.: An experimental comparison of four graph drawing algorithms. Computational Geometry: Theory and Applications **7** (1997) 303–325
11. E.R. Gansner and E. Koutsifios and S.C. North and K.P. Vo: A Technique for Drawing Directed Graphs. IEEE Trans. Software Engg. **19** (1993) 214–230
12. Chimani, M., Gutwenger, C., Mutzel, P., Wong, H.M.: Layer-free upward crossing minimization. In: Proc. 7th Int. Workshop on Experimental Algorithms. Volume 5038 of LNCS. (2008) 55–68
13. Stallmann, M., Brglez, F., Ghosh, D.: Heuristics, Experimental Subjects, and Treatment Evaluation in Bigraph Crossing Minimization. Journal on Experimental Algorithmics **6**(8) (2001)
14. Watson, B., Brink, D., Stallmann, M., Rhyne, T.M., Devarajan, R., Patel, H.: Visualizing very large layered graphs with quilts. Technical Report 17, North Carolina State University (2008)
15. Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Transactions on Systems, Man, and Cybernetics **11** (1981) 109–125
16. Eades, P., Wormald, N.C.: Edge Crossings in Drawings of Bipartite Graphs. Algorithmica **11** (1994) 379–403
17. Li, X.Y., Stallmann, M.F.: New bounds on the barycenter heuristic for bipartite graph drawing. Information Processing Letters **82** (2002) 293–298