Using Sifting for k-Layer Straightline Crossing Minimization

Christian Matuszewski, Robby Schönfeld, and Paul Molitor

Institute for Computer Science, University Halle-Wittenberg,
Kurt-Mothes-Strasse 1, D-06120 Halle(Saale), Germany
{matuszew,schoenfe,molitor}@informatik.uni-halle.de

Abstract. We present a new algorithm for k-layer straightline crossing minimization which is based on sifting that is a heuristic for dynamic reordering of decision diagrams used during logic synthesis and formal verification of logic circuits. The experiments prove sifting to be very efficient. In particular it outperforms the traditional layer by layer sweep based heuristics known from literature by far when applied to k-layered graphs with $k \geq 3$.

1 Introduction

Directed graphs are commonly used to represent information in many fields such as software engineering, project management, and social sciences. A good visualization of this information is necessary to get general ideas of relations.

A common method for drawing directed graphs was introduced by [STT81]. This approach consists of four phases. In the first step, the directed graph is made acyclic by temporarily reversing some edges. Let $G = (V, E)$ be the resulting graph. Then, set V of the nodes is partitioned into k layers $V_1, \ldots, V_k$ $(k \geq 1)$ such that for any edge $(u, v) \in E$ with $u \in V_i$ and $v \in V_j$, the inequation $i < j$ holds. Subset V_i is called ith layer. $G = (V_1 \cup \ldots \cup V_k, E)$ is called a k-layered directed graph. In the third step, the vertices within each layer are permuted targeting minimization of the number of crossings. In the last step, the vertices and edges are placed according to the permutations computed in step 3. The vertices of each layer are placed on a horizontally line, the layer i is placed above the layer $i+1$ for all $i = 1, \ldots, k-1$, and the edges are drawn as straight lines.

In this paper, we focus on step 3. We assume that graph G has been made proper in step 2, i.e., that for any edge $(u, v) \in E$ with $u \in V_i$ and $v \in V_j$ the equation $j - i = 1$ holds. This can be achieved by inserting dummy vertices on edges connecting vertices on non-neighboring layers. The problem we have to solve in step 3 is to find a permutation π_i for every layer V_i such that the number of edge crossings is minimized. Note that the number of edge crossings only depends on the permutations of the vertices and not on the exact positions of the vertices because edges are drawn as straight lines. Unfortunately, the problem of finding a permutation π_i for every layer V_i such that the number of crossings is minimal is NP-hard even for 2-layered directed graphs [GJ83] which

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 217–224, 1999.

we call *two sided crossing minimization problem* in the following. Furthermore, the problem of 2-layer straightline crossing minimization remains NP-hard when the permutation of one layer is fixed [EW94a]. We call this problem the *one sided crossing minimization problem*.

In Section 2 we shortly review heuristics known from the literature. Section 3 presents the new approach which is a standard technique for minimization of binary decision diagrams (BDDs) during logic synthesis and formal verification of logic circuits. The experimental procedure of this section only concentrates on the one sided crossing minimization problem. We generalize sifting to general k-layer straightline crossing minimization in Section 4.

2 Algorithms Known from the Literature

Most methods to reduce the crossings in a k-layered directed graph use a technique called the *layer-by-layer sweep* which works as follows. First, an initial vertex ordering for every layer is computed. Then, for $i = 2, \ldots, k$, the vertex ordering of layer V_{i-1} is held fixed while the vertices of layer V_i are reordered to reduce crossings. After that, the method sweeps back holding fixed the permutation of layer V_i and permute the vertices in layer V_{i-1}. This process is repeated until no further refinement can be achieved. With this approach the k-layer crossing minimization problem is reduced to a series of one sided crossing minimization problems.

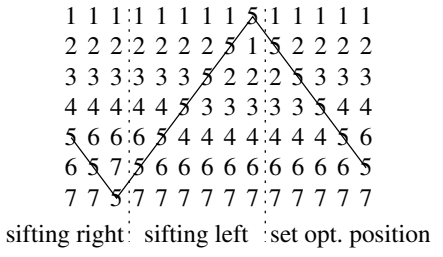
Let us consider the one sided crossing minimization problem in more detail. Without loss of generality, let the permutation π_1 of layer V_1 be fixed. For every pair of vertices $u, v \in V_2$ we define the *crossing number* c_{uv} as the number of edge crossings that edges incident to u make with edges incident to v if $\pi_2(u) < \pi_2(v)$ holds. Thus, $\text{cross}(G, \pi_1, \pi_2) = \sum_{\pi_2(u)<\pi_2(v)} c_{uv}$ is the number of crossings in the straightline drawing of the 2-layered graph $G = (V_1 \cup V_2, E)$ with respect to the permutations π_1 and π_2. An obvious lower bound of $\text{cross}(G, \pi_1, \pi_2)$ is given by $L = \sum_{\pi_2(u)<\pi_2(v)} \min\{c_{uv}, c_{vu}\}$. Experiments in [JM97] have proven this lower bound to be very tight to the optimum.

The most popular heuristics for one sided crossing minimization are the barycenter [STT81] and the median heuristic [EW94b]. The other heuristics known from literature, e.g., split [EK86], greedy-insert [EK86], and greedy-switch heuristic [EK86], are mostly outperformed by barycenter and median heuristic as shown by [JM97]. It is worth remarking that Jünger and Mutzel presented a very efficient branch and cut algorithm for one sided crossing minimization practically applicable to graphs up to 60 vertices in their paper.

3 Sifting for One Sided Crossing Minimization

Sifting was first introduced by Rudell [Rud93] to reduce the number of vertices in reduced ordered binary decision diagrams (ROBDDs). This algorithm can be easily adapted to the one sided crossing minimization problem. Assume that

permutation π_1 of layer V_1 is fixed. We choose a vertex v from layer V_2 to put it on a position which minimizes the number of crossings. The ordering of all other vertices in layer V_2 remains fixed. In order to do so, vertex v is moved to the rightmost position by repeatedly swapping it with its right neighbor. After reaching the rightmost position, v is moved to the left. When v reaches the leftmost position, it is set to the (locally) optimal location. For illustration, please see Figure 1 on the left.



$failcount \leftarrow 0$
repeat
  foreach $v_i \in \phi$ **do** `sifting`(v_i)
  if *fail* **then**
    `reverse`(ϕ)
    $failcount \leftarrow failcount + 1$
  foreach $v_i \in \phi$ **do** `sifting`(v_i)
  if *fail* **then**
    $failcount \leftarrow failcount + 1$
  `reverse`(ϕ)
until $failcount \geq MAXFAILS$

Fig. 1. `Sifting` of vertex 5 (left) and `Global Sifting` (right).

To find the best position for a given vertex, we have to compute the number of crossings after each swapping. Assume we have to swap the two vertices u and v with u being the left neighbor of v. Then, after the swapping step there are $\text{cross}_{after\_swapping} = \text{cross}_{before\_swapping} - c_{uv} + c_{vu}$ crossings. With that, sifting of one vertex takes $\mathcal{O}(n)$ resulting in an overall runtime of $\mathcal{O}(n^2)$, where n is the number of vertices.

We consider three different methods for choosing the next vertex to be sifted. Method 1 uses a preassigned vertex order of the layer, e.g., from left to right. A more accurate approach seems to be to sift the vertices according to their degrees, i.e., sift the vertices with high indegree, first (method 2). The third method investigated by us randomly chooses the vertices.

Experimental Results for one Sided Crossing Minimization

Our computational experiments include the barycenter heuristic, the median heuristic, the greedy-insert heuristic, the greedy-switch heuristic, the split heuristic, and the sifting heuristics (method 1 up to method 3). For our experiments we used the algorithms implemented in the AGD library [MGB+98] which is based on LEDA [MN99]. We have used the program `random_bigraph` of the Stanford GraphBase [Knu93] with the same parameters as chosen by Jünger and

Mutzel [JM97]. We could reproduce the results of Jünger and Mutzel exactly except for the median heuristic and the barycenter heuristic. The differences with the median heuristic are due to the fact that the implementation in AGD uses the *average median* from [Mäk90] rather than the original median heuristic from [EW94b]. Our results are always better than those given in [JM97]. The slight deviation from the results with the barycenter heuristic are possibly due to minor differences in sorting methods.

In the first experimental run for one sided crossing minimization, we have considered sparse graphs with an increasing number of vertices, 10 samples for each type of graphs. Such instances are among the most interesting in practical applications. We have used sparse graphs with $|E| = |V_1| + |V_2|$, i.e., on the average, two edges are incident to each vertex. Figure 2 gives the relative size of the average number of crossings taken over all sampled instances of the given graph type in percentage of the minimum number of crossings, which has been computed by the branch and cut algorithm for one sided crossing minimization of Jünger and Mutzel [JM97]. The curves prove that all three sifting methods are close to the optimum for sparse graphs and that sifting dominates the heuristics known from literature for that class of graphs. The running time of the sifting algorithm is below 1 second [1] for 100+100-graphs, i.e., graphs $G = (V_1 \cup V_2, E)$ with $V_1 = V_2 = 100$.

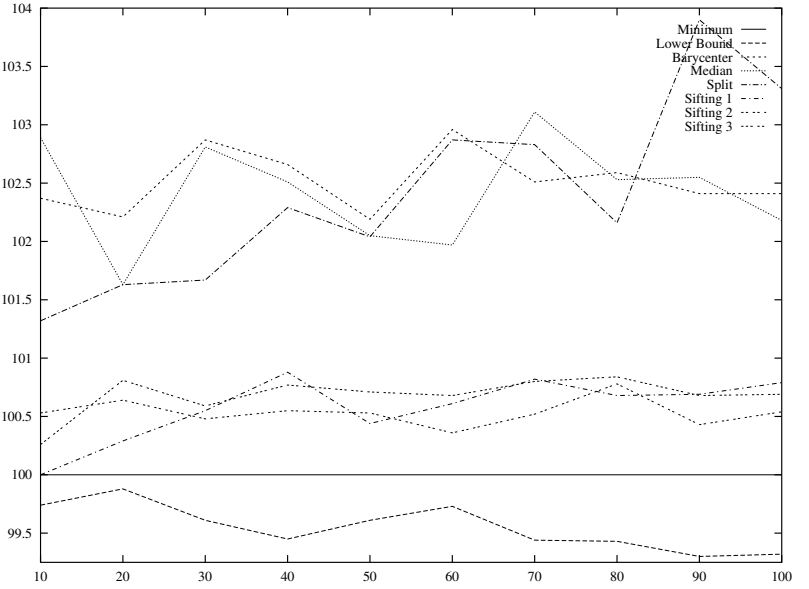


Fig. 2. Results for one sided crossing minimization, sparse graphs, increasing number of vertices, 10 samples.

[1] 296 MHz SUNW, UltraSPARC-II, 512 MB.

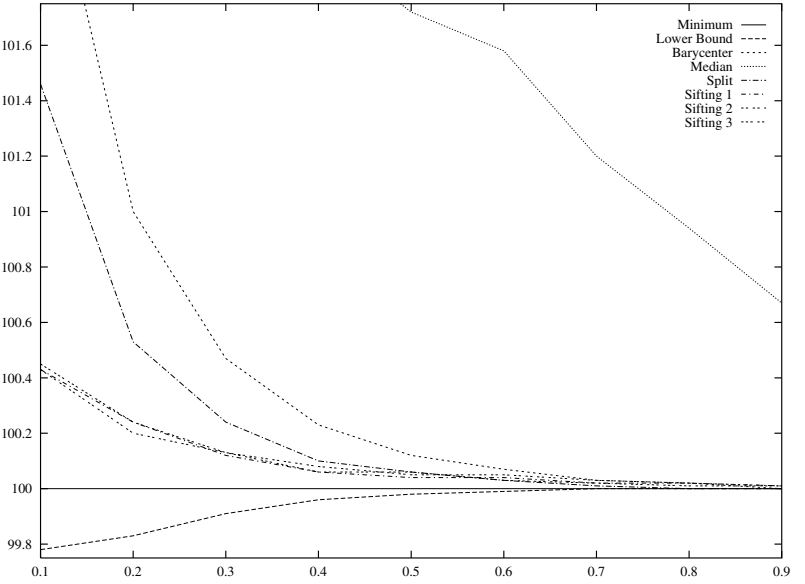


Fig. 3. Results for one sided crossing minimization, 20+20-graphs, increasing edge density, 100 samples.

The next experimental run we have made considers 20+20-graphs with increasing edge densities from 10 up to 90 percent. Figure 3 gives the result for 100 samples for each type of graph. It shows that for one sided crossing minimization sifting dominates the other heuristics for small densities. In the case of high densities, the barycenter heuristic and the split heuristic are as efficient as sifting is.

To sum it up it can be said that sifting applied to one sided crossing minimization is a very efficient heuristic and the vertex order used during sifting does not influence the quality of the heuristic very much.

4 Extending Sifting to k-Layered Directed Graphs

As stated in [JM97], there is no real need for heuristics for one sided crossing minimization up to 60 vertices in the permutable layer as the optimum solution can be computed fast by the branch and cut algorithm. So, we should concentrate on the general problem, namely on the k-layer straightline crossing minimization problem for $k \geq 2$.

Sifting can obviously be extended to k-layer straightline crossing minimization by using layer by layer based sifting (see Section 2). Another method which we call `Global Sifting` is to keep a list $\phi = (v_1, \ldots, v_n)$ of the vertices of V in descending order of the degree of the vertices and to sift each vertex in its associated layer according to this order. Vertices with high degree are handled, first.

If no improvement is made, we reverse the vertex order of list ϕ and memorize it as a fail , otherwise we use the same sequence for the next trial again. After these two loops passed, the vertex order is inverted in any case and the cycle is repeated if the number of fails is less than a defined maximum count. Despite the possibility of better results, we have not taken care about this constant and set it to zero in all experimental runs. Note that in this algorithm optimization is not only realized layer by layer. Figure 1 on the right presents the algorithm in detail.

Experimental Results for k-Layered Graphs

In our experiments, we have considered directed graphs up to 12 layers with different numbers of vertices and edge densities. All heuristics, except global sifting, are iterated between the k layers until a local optimum is obtained.

For small sparse 2-layered graphs $G = (V_1 \cup V_2, E)$ with $|E| = |V_1| + |V_2|$ sifting is slightly better than the barycenter method, whereas barycenter dominates sifting for larger sparse 2-layered graphs. The insufficient quality of sifting for increasing number of vertices seemed to be due to the fact that the density decreases when the number of vertices increases. A sparse 10+10-graph, e.g., contains 20 edges resulting in a density of 20 percent, whereas a 100+100-graph which contains 200 edges leads to a density of only 2 percent. We repeated our experimental run for increasing number of vertices with a constant edge density of 20 percent. The results gave evidence of our assumption. Sifting and global sifting are slightly better than the barycenter method in this case. However, this slight improvement does not justify the higher computation time of sifting.

For more than 2 layers, global sifting dominates all the other heuristics by far. This fact is illustrated by Figure 4 and 5. Here, the horizontal curve at the 100 percentage level represents the results for global sifting. In the experiments, we used k*b-graphs, i.e., k-layered directed graphs $G_k = (V_1 \cup \ldots \cup V_k, E)$ with $|V_i| = b$ for all $1 \leq i \leq k$.

In a first experimental run, we varied the number of vertices in sparse graphs with 4 layers. For this, we extended the concept of sparse graphs from 2 layers to k layers by only considering k*b-graphs $G = (V_1 \cup \ldots \cup V_k, E)$ with $|E| = 2 \cdot (k-1) \cdot b$. Then, on the average, a vertex must have 2 neighbors in each adjacent layer. Figure 4 shows the results for sparse k*b-graphs with $k = 4$ and $10 \leq b \leq 100$ taken over 10 samples. Surprisingly, layer by layer based sifting is much worse than the barycenter, median, and split heuristic. However, global sifting is very efficient and outperforms all the other methods.

The next experimental run, we have made, considers k*b-graphs with higher densities. Although all the heuristics converge to the true optima when the edge density is increased, global sifting dominates the other heuristics. In particular it is much more efficient for dense graphs than all the other heuristics, too. Detailed information on this experiment are included in the technical report appeared at Martin-Luther-University.

Finally, we studied the influence of the number of layers on the quality of the crossing minimization (see Figure 5). For this, we have used sparse graphs with 10 vertices per layer. The number of layers is rising from 2 up to 12 layers. It

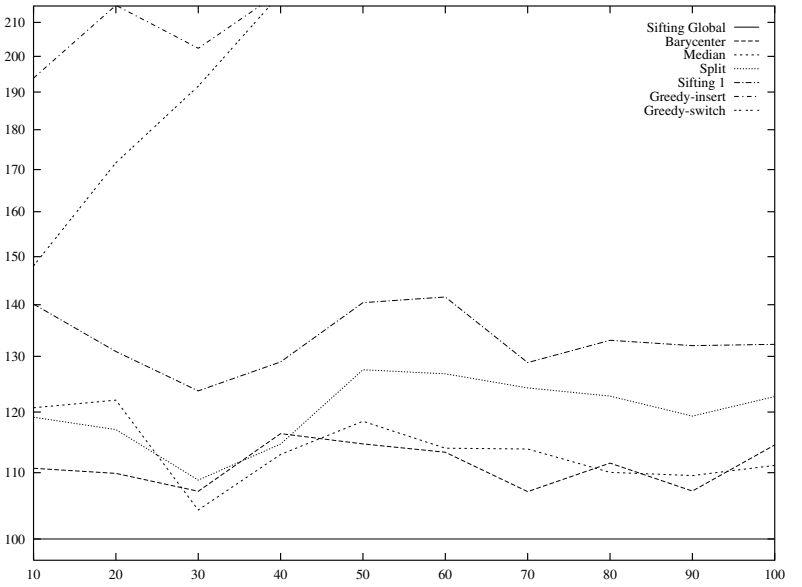


Fig. 4. Results for 4-layer crossing minimization, sparse graphs, increasing number of vertices, 10 samples.

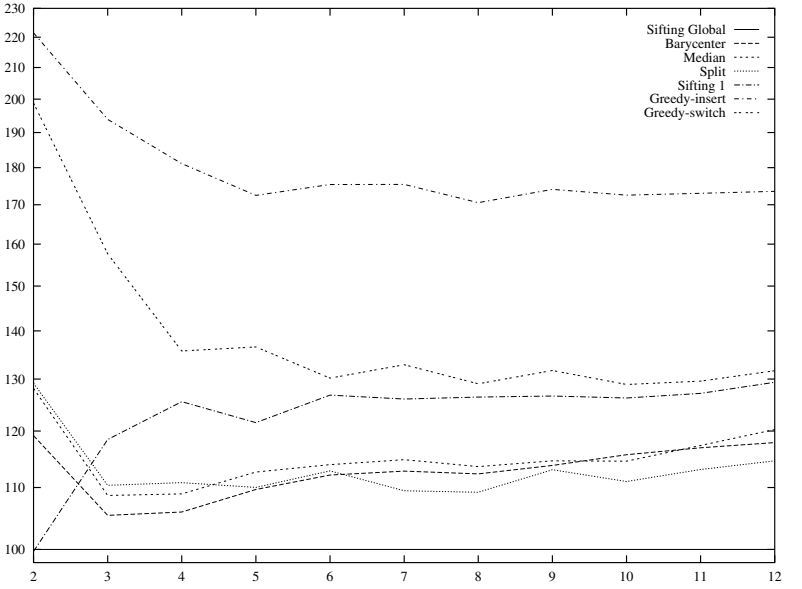


Fig. 5. Results for k-layer crossing minimization, sparse graphs, 10 vertices per layer, increasing number of layers, 100 samples.

can be observed that the layer by layer sweep based heuristics lose performance against global sifting for increasing number of layers.

5 Conclusions

We have presented a new approach for k-layer straightline crossing minimization which has been proven to be efficient. The experiments lead to the conclusion that global sifting dominates the heuristics known from literature for one sided and k-layered crossing straightline minimization with $k \geq 3$.

Acknowledgments

We would like to thank Petra Mutzel, Thomas Ziegler, and Stefan Näher for helpful discussions concerning the AGD library and LEDA.

References

EK86. P. Eades and D. Kelly. Heuristics for reducing crossings in 2-layered networks. *Ars Combin.*, 21.A:89–98, 1986.

EW94a. P. Eades and S. Whitesides. Drawing graphs in two layers. *Theoretical Computer Science*, 131:361–374, 1994.

EW94b. P. Eades and N. C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11(4):379–403, 1994.

GJ83. M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 4:312–316, 1983.

JM97. M. Jünger and P. Mutzel. 2-Layer straightline crossing minimization: Performance of exact and heuristic algorithms. *J. Graph Algorithms Appl.*, 1(1):1–25, 1997.

Knu93. D. E. Knuth. *The Stanford GraphBase: A Platform for Combinatorial Computing*. Addison-Wesley, Reading, MA, 1993.

Mäk90. E. Mäkinen. Experiments on drawing 2-level hierarchical graphs. *Internat. J. Comput. Math.*, 36:175–181, 1990.

MGB+98. P. Mutzel, C. Gutwenger, R. Brockenauer, S. Fialko, G. W. Klau, M. Krüger, T. Ziegler, S. Näher, D. Alberts, D. Ambras, G. Koch, M. Jünger, C. Buchheim, and S. Leipert. A library of algorithms for graph drawing. In S. H. Whitesides, editor, *Proceedings of the 6th International Symposium on Graph Drawing (GD '98)*, volume 1547 of *Lecture Notes in Computer Science*, pages 456–457. Springer, 1998. Project home page at `http://www.mpi-sb.mpg.de/AGD/`.

MN99. K. Mehlhorn and S. Näher. *The* LEDA *Platform of Combinatorial and Geometric Computing*. Cambridge University Press, 1999. Project home page at `http://www.mpi-sb.mpg.de/LEDA/`.

Rud93. R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proc. International Conf. on Computer-Aided Design (ICCAD)*, pages 42–47, November 1993.

STT81. 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, February 1981.