Orthogonal Hypergraph Routing for Improved Visibility

Thomas Eschbach
Wolfgang Günther
Bernd Becker

Institute of Computer Science
Albert-Ludwigs-University
Georges-Köhler-Allee 51
79110 Freiburg im Breisgau, Germany

Report 192, August 2003

Orthogonal Hypergraph Routing for Improved Visibility

T. Eschbach[1] W. Günther[2] B. Becker[1]

[1] Institute of Computer Science
Albert-Ludwigs-University
79110 Freiburg im Breisgau, Germany
{eschbach,becker}@informatik.uni-freiburg.de

[2] Infineon AG
CL DAT TDM VM
81730 Munich, Germany
wolfgang.guenther@infineon.com

Abstract

Visualization of hypergraphs is an important research area in electronic design and automation. One commonly accepted method to visualize a circuit places all gates in several layers and then routes the wires in an orthogonal way between the layers. Our approach introduces the restriction that every net is allowed to occupy only one track. This avoids unnecessary bends in the wires and helps to improve the clarity of the drawn circuit. Then a crossing reduction step is applied to further improve the readability of the circuit schematics.

Assume that some elements have already been fixed on a layered graph structure. In this paper, we consider the problem of assigning the hyperedges between two layers to tracks to minimize the total number of hyperedge crossings. First, we prove that solving this problem optimally is NP-hard. Then, we present a new heuristic to reduce the number of crossings. Finally, some promising experimental results on real circuits are given. The results show that in the majority of cases, the outcome of our algorithm are within 3% of the optimum solution.

1 Introduction

The number of signal crossings and the number of bends are commonly accepted measures for the readability of circuit drawings. Easy-to-read visualizations of circuits are required in the field of circuit design for several reasons. Without trying to give all of them we just want to briefly point out some. Todays highly complex chips are written partly or completely in hardware description languages which then have to be translated into a netlist. However, to obtain a circuit some human interaction is usually required. A fast visualization of the circuit can help to understand the functionality of the design. This is importand in the debugging process. Other applications for circuit visualization include teaching of hardware design, presentation of new synthesis methods, and circuit related documentation.

A common framework in the field of automated circuit visualization splits up the process in several steps:

- The partitioning step divides up the circuit in disjunctive parts all containing only as many elements as fit in a screen or on a sheet of paper [2]. Usually the algorithm tries to minimize the number of hyperedges which are connected to elements of different parts.
- All elements are placed in several layers. The main flow of the infomation processing should point into one direction. Also an even distribution of all elements over the area is preferable. Promising approaches to establish this are given e.g. in [3].
- Next, the circuit is modeled as a graph (e.g. like proposed in [10]). Then, a crossing reduction algorithm is applied to obtain an ordering of all elements in each layer. After that, an algorithm is required to route all the hyperedges between the tracks. In contrast to all methods known from the area of physical design which mainly minimize the number of tracks used, this paper considers an approach that tries to improve the visual clarity of the drawing by only allowing every hyperedge to be part of one track (e.g. see Figure 1). To further improve the visual quality a reordering of the tracks is done to help to reduce the number of edge crossings.

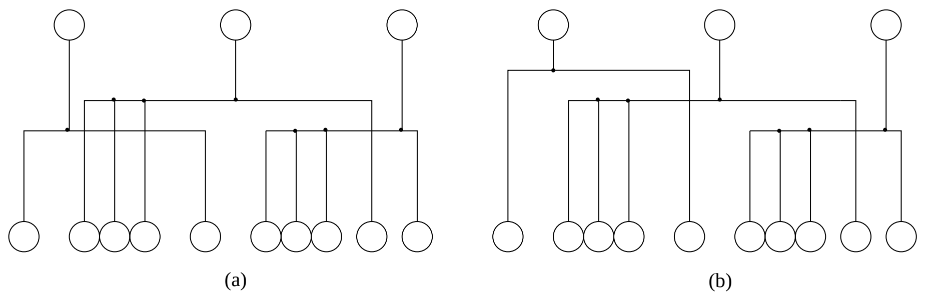

Figure 1: A track minimal embedding (a) does not imply a crossing minimal embedding (b).

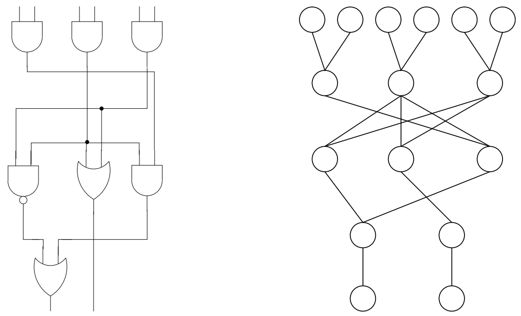

Figure 2: Example of a circuit and its graph representation.

The routing approach for visualization is clearly more convenient for circuit visualization than an approach for minimizing the number of channels since it is easy to follow a single net which is only located in one track. The algorithm also produces less bends and a smaller number of hyperedge crossings than the channel router which allows the hyperedges to spread over several tracks.
Since the computational complexity of the problem has not been investigated before, theoretical results are of large interest to find some efficient algorithms. Unfortunately, it turned out that the problem of assigning the hyperedges to the tracks in an optimum way is NP-hard. Therefore it is unlikely that there exists an algorithm which computes the optimum result in polynomial time.
However, one can find good solutions using the proposed heuristic methods which are able to cope with large instances of the problem in a short period of time.
Experimental results demonstrate that this approach is very efficient, even on large circuits, since the presented heuristic produces on average only about three percent more hyperedge crossings compared to a given lower bound.

This paper is structured as follows: First, some definitions on hypergraphs and graphs are given. Then a proof is provided to show that the decision problem on hyperedge crossing minimization is NP-complete. Next a new heuristic method to minimize the number of crossings is presented. Finally, some experimental results are published and summarized with a short conclusion.

2 Preliminaries

A circuit is a hypergraph $H = (V, E)$ where the gates are represented by nodes V and the nets are represented by hyperedges (subsets of V). Each hyperedge consists of all the wires which are directly connected to each other. The circuits are modeled as proposed in [10] to obtain a directed graph. This is done by creating a node for every input, every output and every gate of the circuit. Then, the algorithm inserts the corresponding $|E| - 1$ edges for every hyperedge E. An example of a circuit and its graph representation are given in Figure 2.

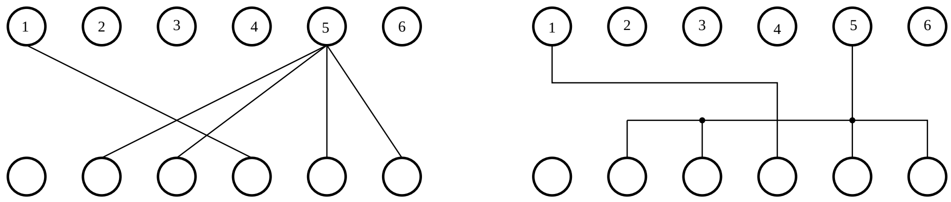

Figure 3: One example graph and the corresponding orthogonal embedding.

A directed graph $G = (V, E)$ is a *multi-layered* graph with d layers if the node set V is partitioned into disjoint subsets V_1, V_2, …, V_d, i.e. $V_1 \cup V_2 \cup \ldots \cup V_d = V$ and $(\forall m \neq m')$ $V_m \cap V_{m'} = \emptyset$. V_m is called the *m-th layer* of the graph. All edges in E connect nodes in different layers. A layering of a graph is called a *proper* layering if the edges are only connected to nodes of adjacent layers V_m and V_{m+1}. If a layering of a graph is not a proper layering, one can introduce *dummy nodes* along edges (u, v) for which $l = \text{layer}(v) - \text{layer}(u) > 1$. We replace (u, v) by a path $(u = v_1, v_2, \ldots, v_l = v)$ of length l. In each layer between u and v, one dummy node has to be inserted. From now on we assume that the graphs are proper layered.

It is important to note that the order of the nodes in a layer V_m, $m \in \{1, \ldots, d\}$, only affects the number of crossings with adjacent layers. To solve the exact multi-layer straight-line crossing minimization problem we have to determine an order ord_m for all layers m containing all the nodes in layer V_m so that the number of crossings is minimized. In the following a set of orders ord_m, $m \in \{1, \ldots, d\}$, is called an order for the graph. Unfortunately, even minimizing edge crossings in graphs with two layers is NP-hard [8] and remains NP-hard if one ordering of a layer is fixed [5].

One of many known heuristic methods to solve the problem is the averaging heuristic method which was first introduced by [14]. It computes the position of a node n only with respect to all nodes in the above layer which are connected with n. More precisely, it computes the average of the x-coordinates of its neighbors for all nodes on one level. Then it sorts the nodes with respect to this value.

To compute an orthogonal embedding of a circuit the algorithm has to route the wires in the channel between the two layers. For this reason we divide the channel into several tracks. It is important that the viewer of the drawing can easily follow the wires. To make the wiring clearly arranged, one net is only allowed to occupy one track. Therefore, every net is represented by at most one horizontal line and some vertical lines to connect it to all the nodes. It is easy to see that in this case, for a given ordering of the nodes the number of bends in each channel is also very low. This strategy is in contrast to many other routing algorithms which only try to minimize the number of tracks. For an example see Figure 1.

We expand the notation of the crossing number [4] introduced for layered graphs on layered circuits. For each pair of nets we define the crossing number c_{ij} as the number of crossings between the nets i and j where net i is assigned to a higher track than net j. Furthermore, we define $c_{ii} = 0$ for all nets. On the right side of Figure 3, the crossing number c_{15} between net 1 and net 5 is defined as one and the crossing number c_{51} is defined as two.

An orthogonal embedding can reduce the number of crossings compared to the straight line embedding of its corresponding graph as shown in Figure 3. If π is the actual permutation of the nets in the channel, one can compute the number of caused crossing C as

$$C = \sum_{\pi(i)<\pi(j)} c_{ij}.$$

Also, a lower bound L on the number of crossing can be computed with

$$L = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} min\{c_{ij}, c_{ji}\}.$$

It is easy to see that the minimal number of crossings can be larger than L. However, if for one given permutation the actual number of crossings matches the lower bound the algorithm has found an optimal embedding.

3 Hypergraph Crossing Minimization is Difficult

Unfortunately it turned out that computing an optimal embedding of all hyperedges in the given framework is NP-hard. In the reminder of this section we give a a proof of the claimed proposition.

Hypergraph crossing minimization (HCM) can be transformed into the decision hypergraph crossing problem (DHCP). Given a hypergraph $H = (L_0, L_1, h)$ with two layers L_0 and L_1, a set of hyperedges h, an ordering π_0 of L_0, an ordering π_1 of L_1, and an integer $\tilde{C}$, is there an ordering π_h of all hyperedges h so that the number of hyperedge crossings is $\leq \tilde{C}$?

It is easy to see that the DHCP problem is in NP. To show that the DHCP problem is NP-complete, we transform the feedback arc set into a DHCP problem. As shown in [9, 7] the problem of finding a subset A' of arcs from a given graph $G = (V, A)$ where $|A'| \leq k$ such that A' contains at least one arc from every directed cycle in G is NP-complete.
Let the graph $G = (V, A)$ together with the positive integer $k \leq |A|$ constitute an arbitrary instance of the feedback arc set problem. Let $n = |V|$ be the number of nodes of the feedback arc set problem. We now demonstrate how each instance of the feedback arc set problem can be transformed in polynomial time to the corresponding DHCP problem in the following way:

First, for every node of the feedback arc set problem, one node is placed on layer L_0 and two nodes are placed on layer L_1 of the hypergraph. For the i-th node of the graph, we place one node located on L_0 on position $n-i+1$ and two nodes located on L_1 on position i and $n+i$. Without loss of generality all nodes located on L_0 are placed in the middle of the drawing, more precisely between nodes n and $n+1$ of layer L_1 (see also Figure 4). The vertical dotted lines in Figure 4 are only drawn to show the $2n-1$ different parts of the hypergraph, which are introduced if the hypergraph is constructed in this manner.

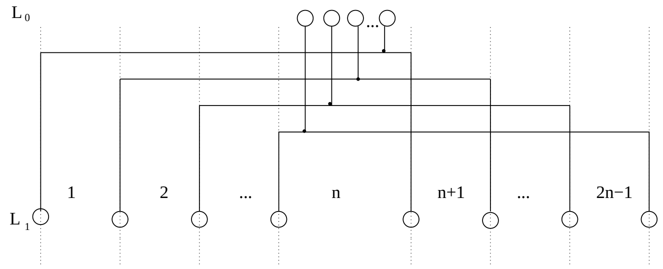

Figure 4: Transformation of the nodes.

Every hyperedge of this graph uses only one horizontal track. Two hyperedges have exactly two crossings with each other which leads to $2*((n-1)+(n-2)+(n-3)+\ldots+(n-(n-1)))$ different hyperedge crossings. It is important to notice that because of the symmetry of the arrangement the number of hyperedge crossings is independent of the assignment of hyperedges to horizontal tracks since every arbitrary permutation of two hyperedges in adjacent tracks does not change the total number of hyperedge crossings. Therefore, without loss of generality, we use the simplified representation of the constructed hypergraph shown in
Figure 5 for the rest of the proof.

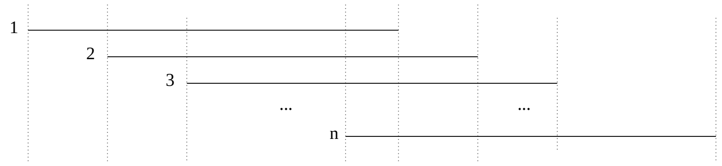

Figure 5: Transformation from a graph with n nodes without arcs.

Now, for every arc of the feedback arc set problem, we introduce a corresponding couple of nodes on layer L_1, which are represented with vertical lines in our simplified representation of the hypergraph. In general, an arc in the feedback arc set problem can point from a node with a smaller number to a node with larger number, or vice versa.
In the first case, let the arc point from node x to node y where $x < y$. For this arc one additional vertical line is added to hyperedge x in the $(y-1)$'th part of the hypergraph. Next, $y-1$ vertical lines are added in the y'th part of the hypergraph connected to all the hyperedges except hyperedge x. In Figure 6, a transformation of a graph with five nodes and one arc pointing form node 2 to node 4 is given.

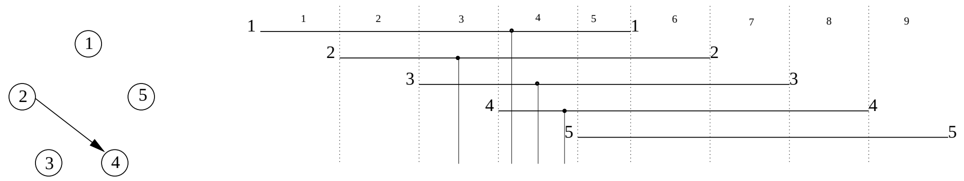

Figure 6: A graph with five nodes and one arc from node 2 to node 4 and its hypergraph.

Please notice that the added vertical lines induce a constant number of hyperedge crossings c_{xy} for all permutations of the hyperedges when x is located above y since the additional vertical lines induce one hyperedge crossing for every pair of involved hyperedges except for the pair x and y (see also Figure 6). Also, all permutations of the hyperedges induce exactly $c_{xy} + 1$ hyperedge crossings when y is located above x.

In the second case where the arc points form node x to node y and $x > y$, one additional vertical line is connected to hyperedge y in the x'th part of the hypergraph. Then $x - 2$ vertical lines are connected to all hyperedges in the $(x-1)$'th part of the hypergraph except hyperedge y. Again, the added vertical lines are inducing a constant number of hyperedge crossings c_{xy} for all permutations of the hyperedges where x is located above y. For all permutations of the hyperedges where y is located above x, the added vertical lines induce exactly $c_{xy}+1$ hyperedge crossings. In Figure 7, a transformation of a graph with five nodes and one arc pointing form node 4 to node 2 is given.

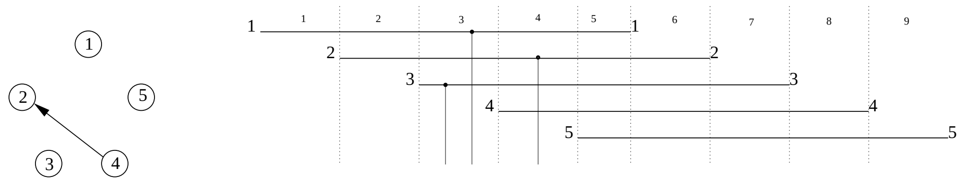

Figure 7: A graph with five nodes and one arc from node 4 to node 2 and its hypergraph.

Let C be $\sum_{(x,y)\in A} c_{xy} + 2 * ((n-1) + (n-2) + (n-3) + \ldots + (n-(n-1)))$. C is a lower bound to the hyperedge crossing minimization problem since in π_h every hyperedge h_i has to be placed before or after every hyperedge h_j ($i \neq j$ and $i, j \in 1, 2, \ldots, n$).
Now we show that G has a feedback arc set of size $\leq$ k if and only if an ordering of the hyperedges can be chosen so that the number of hyperedge crossings is $\leq \tilde{C} = C + k$.

If the graph has a feedback arc set A' of size $\leq k$ the resulting acyclic graph $G' = (V, A - A')$ can be used to obtain an ordering $\pi_{G'}$ of G' by a topological sort. This topological ordering assures that $\pi_{G'}(x) < \pi_{G'}(y)$ for all arcs $(x,y) \in A - A'$. Now the same ordering is used to assign the hyperedges to the tracks. This implies that each arc $\in A'$, where $A' = \{(x,y) \in A : \pi_{G'}(x)$

$> \pi_{G'}(y)\}$ produces one additional hyperedge crossing. Since there are at most k arcs in A' the hypergraph can have at most $\tilde{C} = C + k$ hyperedge crossings.

Otherwise, given an ordering π_h of the hyperedges such that the hypergraph has at most $\tilde{C}$ hyperedge crossings is shown to imply a feedback arc set A' with $|A'| \leq k$. The ordering of the hyperedges π_h is mapped to the ordering of the nodes π_G and then the edge set $A' = \{(x, y) \in A : \pi_G(x) > \pi_G(y)\}$ is deleted to obtain an acyclic graph. Because of the construction of the hypergraph, it follows that C hyperedge crossings are caused independent of the ordering π_h and at most k additional hyperedge crossings are caused by the set $A' = \{(x, y) \in A : \pi_G(x) > \pi_G(y)\}$. Furthermore $|A'| \leq k$ and the graph $G' = (V, A - A')$ is acyclic, therefore, A' is a feedback arc set.

4 Greedy and Sifting Heuristic Method

The computational complexity of the hyperedge crossing reduction problem requires the usage of fast heuristic methods especially in the field of circuit visualization where a short respond time is crucial. The next sections briefly describe the greedy and the sifting heuristic method.

4.1 Greedy Assign

The greedy assign heuristic method iteratively assigns hyperedges starting at the top most track. Next the algorithm computes the number of crossings for all hyperedges which are induced if the hyperedge is positioned above the remaining hyperedges. Then it chooses the hyperedge which causes the smallest number of crossings and fixes it to the top most track. The algorithm repeats this greedy selection step until all hyperedges are assigned to one track. Pseudo code for the greedy assign method is given below:

```
greedy_assign(crossing_matrix cm, hyperedges nets){
    for k = 1 to |nets| − 1 do {
        mincost = +∞
        for every net i ∈ nets do {
            cost = Σ_{j∈nets} c_ij
            if(cost < mincost) {
                best = i
                mincost = cost
            }
            nets = nets \ best
        }
        assign best to track k
```

```
    }
}
```

4.2 Sifting

The sifting algorithm [13] chooses one net and moves it to the lowest track by repeatedly swapping it with its neighbor. Next, the chosen nets is continuously swapped from the lowest track up to the topmost track. Finally, it is moved to its locally optimal position. The algorithm then proceeds with the next net until every net has been touched. It is easy to see that swapping a net with its neighbor is a local operation and that the number of crossings after the operation is

$$Crossings = Crossings_{before} - c_{ij} + c_{ji}.$$

Usually the algorithm is repeated until no further improvement of the solution can be achieved. The algorithm can be improved in many ways. For further details we refer to [6]. The sifting of one net takes $O(n)$ time, and sifting all nets summarizes to $O(n^2)$ time.

5 Experimental Results

The algorithm has been implemented in C and was integrated into AGD [12]. All experimental results are based on examples which were taken from the sequential benchmark circuits in [1, 11]. The experiments were carried out on a 2 GHz personal computer with 1 GB main memory running the linux OS.

We utilized the AGD averaging implementation combined with the greedy switch method to obtain a fast embedding of the nodes. The greedy approach was applied to assign each hyperedge to one track in the corresponding channel. Then the sifting heuristic method is applied to further reduce the number of crossings. This is done to get the final orthogonal layout of each channel with respect to the number of crossings.

In Table 1 the results are summarized. The number of nodes is given in the second column. In the third column the number of straight line crossings can be found. In the forth column we show the number of hyperedge crossings produced by a randomly generated permutation of the hyperedges over the tracks. In the fifth column we present results in terms of hyperedge crossing for the sifting implementation postprocessing a randomly generated starting permutation. In the sixth column we publish the number of hyperedge crossings obtained only using the greedy assign heuristic method. In the seventh column the final results in terms of hyperedge crossing computed with the combination of both methods are shown. Please notice that the sifting heuristic method has very limited possibilities to overcome a local minima and therefore the quality of the solution is very sensitive to a good starting permutation. In the last column the lower bound for every

Table 1: Benchmark results

circuit	nodes	straightline	random	sifting	greedy	sift+gre	lowerb
add6	504	170	706	217	169	158	142
alu1	125	91	327	88	81	73	70
alu2	391	558	1466	544	548	481	475
alu3	452	742	1527	722	648	616	605
adr4	210	144	384	129	132	120	115
co14	261	76	513	103	68	63	63
dk17	437	440	718	362	327	306	288
dk27	186	95	207	95	79	74	74
dk48	641	591	1644	593	566	540	520
mish	571	108	1405	128	121	101	101
radd	215	61	201	75	69	64	52
rd53	193	320	521	259	279	245	238
s208	665	280	1159	332	282	264	258
s298	638	1142	3345	928	1052	860	834
s382	903	1199	4479	1040	883	837	822
s386	763	3868	8360	2594	2549	2383	2338
s400	941	1366	4077	779	1007	596	585
vg2	456	157	989	237	130	128	126
x1dn	470	185	1240	250	177	174	167
x9dn	419	188	1089	224	176	164	164
z4	238	149	414	139	133	124	117
Z9sym	767	5189	8230	3692	3748	3467	3385
$\sum$		17119	43001	13530	13224	11838	11539

circuit is provided. It turned out that for the considered benchmark circuits the greedy assign heuristic method combined with the sifting heuristic method is able to compute the optimum solution in terms of hyperedge crossings for some instances. In general only about three percent more hyperedge crossings are obtained using the greedy assign heuristic method combined with the sifting heuristic method compared to the lower bound. The required runtime for the greedy and the sifting heuristic methods are all below 0.01 s per instance and are therefore not shown. Please notice that in almost all cases the number of hyperedge crossings produced from the greedy heuristic method postprocessed with the sifting procedure is significantly smaller than the crossings caused by a straight line embedding.

6 Conclusions

We have presented a new approach that improves the visual clarity of circuit schematics. The new method helps the viewer to easily extract the functionality of the circuit. The improved clarity is achieved by assigning one hyperedge to only one track and then processing it to reduce the number of hyperedge crossings. The fist part of the work provides a polynomial transformation from the feedback arc set problem into the hypergraph crossing reduction problem to prove that the problem is NP-complete. To compute good solutions in an appropriate amount of time we used the new greedy based heuristic method to reduce the number of hyperedge crossing. To further improve the quality, the sifting heuristic method was applied. Some promising results on real benchmarks are given which demonstrate that this framework computes results which are close to the lower bound on almost every instance. Negligible runtime of our algorithms allows the visualization of large circuits in an interactive way.

References

[1] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. *Int'l Symp. Circ. and Systems*, pages 1929–1934, 1989.

[2] R. Drechsler, W. Günther, T. Eschbach, L. Linhard, and G. Angst. Recursive bi-partitioning of netlists for large number of partitions. *Euromicro Symposium on DSD*, pages 38–44, 2002.

[3] R. Drechsler, W. Günther, L. Linhard, and G. Angst. Level assignment for displaying combinational logic. *Euromicro Symposium on DSD*, pages 148–151, 2001.

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

[5] P. Eades and N. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11:379–403, 1994.

[6] T. Eschbach, W. Günther, and B. Becker. Crossing reduction for orthogonal circuit visualization. *Proceedings of the 2003 International Conference on VLSI, Las Vegas, USA, CSREA Press*, 2003.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NPCompleteness*. Freeman, 1979.

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

[9] R.M. Karp. *Reducibility among combinatorial problems, in Complexity of Computer Computations (R.E. Miller and J.M. Thatcher, eds.)*. Plenum Press, 1972.

[10] E. Mäkinen. How to draw a hypergraph. *International Journal of Computer and Mathematics*, 34:177–185, 1990.

[11] K. McElvain. Benchmark set: Version 4.0. *International Workshop on Logic Synthesis*, 1993.

[12] 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. AGD: A library of algorithms for graph drawing. *Proceedings of the 6th International Symposium on Graph Drawing*, LNCS 1547:456–457, 1998.

[13] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. *ICCAD*, pages 42–47, 1993.

[14] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transaction on Systems, Man and Cybernetics*, 11(2):109–125, 1981.