

## Orthogonal Hypergraph Drawing for Improved Visibility

*Thomas Eschbach*

Albert-Ludwigs-University  
Freiburg im Breisgau, Germany  
<http://ira.informatik.uni-freiburg.de/>  
[eschbach@informatik.uni-freiburg.de](mailto:eschbach@informatik.uni-freiburg.de)

*Wolfgang Günther*

OneSpin Solutions GmbH  
Munich, Germany  
<http://www.onespin-solutions.com>  
[wolfgang.guenther@onespin-solutions.com](mailto:wolfgang.guenther@onespin-solutions.com)

*Bernd Becker*

Albert-Ludwigs-University  
Freiburg im Breisgau, Germany  
<http://ira.informatik.uni-freiburg.de/>  
[becker@informatik.uni-freiburg.de](mailto:becker@informatik.uni-freiburg.de)

### Abstract

Visualization of circuits is an important research area in electronic design automation. One commonly accepted method to visualize a circuit aligns the gates to layers and uses orthogonal lines to connect the gates. In our model we assume that between two consecutive layers every net is allowed to occupy only one track. This avoids unnecessary bends in the wires and helps to improve the clarity of the drawing. Then a crossing reduction step is applied to further improve the readability of the circuit schematics.

First we assume that the nodes have already been fixed on a layered hypergraph structure. We consider the problem of assigning the hyperedges between two layers to tracks. The idea is to minimize the total number of hyperedge crossings. We prove that finding the best solution is NP-hard. Then, in contrast to many other approaches which route all the wiring after placing all nodes we focus on a new approach which dynamically reorders the nodes within the layers to further reduce the number of hyperedge crossings. An efficient algorithm is presented that minimizes the hyperedge crossings. Experimental results are provided which show that the drawings can be improved significantly while the running time remains moderate.

Article Type	Communicated by	Submitted	Revised
regular paper	Y. Kajitani and G. Liotta	April 2004	November 2005

## 1 Introduction

The process of hardware design is divided into several phases. First, a specification of the design is generated which describes the properties of the circuit on a very high level of abstraction. While the first models are quite often written in C, the actual design is usually encoded on the *Register Transfer Level* (RTL) in a special hardware description language like VHDL (see e.g. [3]). In theory, a “real” circuit can be synthesized out of this RTL-model automatically. In practice, however, a large amount of interaction is required for several reasons:

- The generated circuit does not meet some of the requirements. For instance, it has to run at a certain speed, but its delay is too large. Power consumption is another criteria that is becoming increasingly more important. While a modification in the RTL code may improve the performance, sometimes interaction on the gate-level may be required.
- Synthesis tools may translate the RTL model incorrectly. Some variations are widely accepted and common to most synthesis tools because they simplify the compilation process (synthesis-simulation-mismatches), others are simply bugs in the software. While there are powerful industrial tools to find functional differences in two designs (see e.g. [1, 2]), locating the errors still requires an understanding of the design.
- Finally, for various reasons it may be necessary to slightly change the specification late in the design process (Engineering Change Orders, ECOs). To avoid a recompilation of the design, this modification is carried out directly on the circuit’s layout.

In all those cases, an easy-to-read visualization of the circuit is crucial for the understanding of the design. Other applications of circuit visualization include teaching of hardware design, presentation of new synthesis methods, and circuit related documentation.

At first glance the visualization of a circuit and the physical layout of a circuit [21, 27] may seem to be similar problems. However, different goals and different restrictions make it impractical to use existing physical layout algorithms to compute readable circuit schematics. Three important issues are:

- The result of circuit visualization is a two-dimensional drawing, while it is three-dimensional in the physical design process.
- On chip, the area constraints are more important than both the number of crossings and the number of bends. This leads to a compact but confusing routing of the wires.
- Hardware designers are used to a certain “style” of drawings, i.e. lines must be on an orthogonal grid.

A common framework for automated circuit visualization is based on visualization techniques for directed graphs first introduced by [28]. First approaches

to extend the framework to orthogonal hypergraph visualization were given in [12]. An integration into an industrial toolset was discussed in [26]. Theoretical results can be found in [13]. These techniques split the process into several steps:

1. The circuit is partitioned into smaller pieces, either following a hierarchy that is contained in the RTL design, or following a strategy that tries to split the design into parts that can be displayed on a single screen [7]. Depending on the application, this phase may be omitted.
2. The circuit is transformed into a graph. Nodes are used to represent gates, while lines are used to represent wires (see Figure 1).
3. Nodes are assigned to levels. Inputs are put to the top level, outputs to the bottom level [8].
4. Within each level, the nodes are re-ordered such that the number of crossings is minimized [28, 10, 22, 18, 15, 25].

Finally, the graph is transformed back into a circuit which then can be drawn.

It is important to note that the optimization criterion used in the previous phases differs from that of the final drawing: in the graph model, the number of crossings is counted using straight lines, while an orthogonal hypergraph model is used to draw the final circuit. At first glance it is not obvious that optimizing the number of crossings in the first model leads to better solutions in the second model, and there are even examples where the number of crossings differs considerably. Therefore, it makes sense to carry out an additional phase that improves the number of crossings in the final model.

We assume that all the gates are placed on layers, and that they are in a “good” initial order with respect to the orthogonal hypergraph crossing minimization problem. First a proof is provided that finding the best solution is NP-hard. Then, to optimize the order of the horizontal lines, a heuristic approach known from formal verification of hardware is chosen: Sifting [24]. This algorithm was first applied to optimize the variable order in binary decision diagrams (*BDDs*) [5]. *BDDs* are a graph-based data structure that allows efficient representation and manipulation of Boolean functions which are often used in formal verification (e.g. [6]). The algorithm basically works as follows: each horizontal line is chosen one after another. When a line is chosen, all possible positions are examined assuming that the relative order of all the other lines remains the same. Then the line is brought to the position that lead to the smallest number of crossings. The algorithm is iterated until no further improvement is obtained.

Then we extend the framework with a new step which dynamically reorders iteratively two nodes in one layer and repeats the orthogonal embedding of the hyperedges to further reduce the number of hyperedge crossings until a local optimum is reached. This new approach [14] combines the placement and routing process with respect to the hypergraph structure in contrast to many known heuristic methods. Since finding exact solutions for both steps together means

to traverse a huge search space, two fast heuristic methods are used by turns to compute the final solution. Combining two phases during optimization often leads to a significant increase in running time, however, we present experimental evidence that our heuristics do not suffer from this drawback. Experimental results are given which strongly suggest the significant reduction of hyperedge crossings if the new phase is added.

## 2 Preliminaries

A circuit can be modeled as a hypergraph  $HG = (V, H)$  where gates are represented by nodes  $V$  and nets correspond to hyperedges (connecting a subset of  $V$ ). Each hyperedge consists of all wires which are directly connected to each other. Nodes are created for every input, every output and every gate in the circuit. The hypergraph can easily be converted in a directed graph  $G = (V, E)$  by replacing each hyperedge by a set of "corresponding" edges. (For illustration see Figure 2.)

The directed graph  $G = (V, E)$  is used to compute an initial embedding of all nodes in a short time. It can be converted into a *multi-layered* graph with  $d$  layers ( $d \in \mathbb{N}$ ). By this, the node set  $V$  is partitioned into disjoint subsets  $V_1, V_2, \dots, V_d$ , i.e.  $V_1 \cup V_2 \cup \dots \cup V_d = V$  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  $(u, v)$  if  $\text{layer}(v) - \text{layer}(u) > 1$ . We replace  $(u, v)$  by a path of length  $l$  ( $u = v_1, v_2, \dots, v_l = v$ ). In each layer between  $u$  and  $v$ , one dummy node has to be inserted. If multiple dummy nodes belonging to the same hyperedge exist on the same layer, they are merged. Please notice that the dummy nodes are not visualized in the final drawing. An example of a circuit and its graph representation resulting from the above construction is given in Figure 1. From now on we assume that the layering considered is proper.

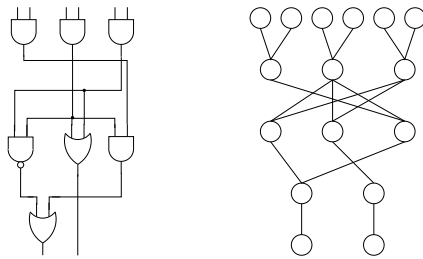


Figure 1: Example of a circuit and its graph representation.

It is important to note that the order of the nodes in a layer  $V_m$ ,  $m \in \{1, \dots, d\}$ , 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. Unfortunately, minimizing edge crossings in graphs with two layers is NP-hard [17], and it remains NP-hard even if one ordering of a layer is fixed [11].

One of many known heuristic methods to solve the problem is the averaging heuristic method [28]. It computes the position of a node  $n$  only with respect to all nodes in the above layer which are connected to it. 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.

We can generalize the notation of the crossing number [10] introduced for layered graphs to layered hypergraphs if the orderings of all nodes are fixed. 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 hand side of Figure 3, the crossing number  $c_{1,5}$  between net 1 and net 5 is defined as one and the crossing number  $c_{5,1}$  is defined as two. It is important to note that the definition of the crossing numbers and all algorithms in this paper can deal with hyperedges which are connected with  $r$  nodes in the upper layer and  $k$  nodes in the lower layer ( $r, k \in \mathbb{N}$ ). However, all experiments are carried out with real benchmark circuits [4, 23] where all hyperedges have a  $1 : k$  relation.

To compute an orthogonal embedding of a circuit, the algorithm has to route the wires in the channel between the two layers. We divide this channel into tracks, and only one wire is allowed per track. This prevents wires within one track from overlapping or being too close. Thus, between two consecutive layers every net is represented by at most one horizontal line and one or multiple vertical lines. This strategy is in contrast to many other algorithms which only try to minimize the number of tracks. For an example, see Figures 2.

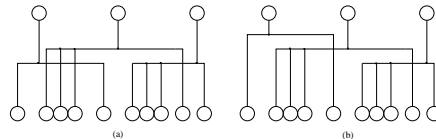


Figure 2: Remark: A track minimal embedding (a) does not imply a crossing minimal embedding (b).

An orthogonal hypergraph embedding can reduce the number of crossings compared to the straight line embedding of its corresponding graph as shown in Figure 3.

If  $\pi$  is the actual permutation of the nets in the channel and the permutation of all nodes is fixed, one can compute the number of crossings  $C$  as

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

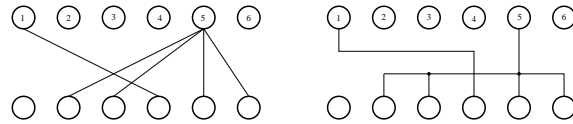


Figure 3: An example graph and the corresponding orthogonal hypergraph embedding.

We now define the hypergraph crossing minimization problem (HCM): Let the following be given: a hypergraph  $H = (L_0, L_1, h)$  with two layers  $L_0$  and  $L_1$ , a set of hyperedges  $h$  which are connected to at maximum one node from layer  $L_0$ , an ordering  $\pi_0$  of  $L_0$ , and an ordering  $\pi_1$  of  $L_1$ . All hyperedges  $h$  are drawn in a orthogonal way and consist of at maximum one horizontal line which is located between the two layers. It is connected to all incident nodes with a vertical line. The HCM problem is now to find an assignment of all hyperedges to different tracks which minimizes the number of hyperedge crossings. In the same way we define the problem of finding an optimal orthogonal embedding of all edges of a graph (GCM).

The complexity results can be directly transfered to all hypergraphs. Restrict the hypergraph to the hypergraphs defined in the HCM problem by only allowing that every hyperedge is at most connected to one node of  $L_0$ .

### 3 Hypergraph Crossing Minimization is Difficult

Computing an optimal embedding of all hyperedges in the given framework is NP-hard. In contrast to that, an optimal solution to the GCM problem in this framework can be computed asymptotically with respect to the number of edges of the given graph. In the reminder of this section we give a proof for both claims.

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  $\pi_0$  of  $L_0$ , an ordering  $\pi_1$  of  $L_1$ , and an integer  $\tilde{C}$ , is there an ordering  $\pi_h$  of all hyperedges  $h$  so that the number of hyperedge crossings is  $\leq \tilde{C}$ ?

**Proof:** 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 it was shown in [20, 16], 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:

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 drawn to show the  $2n - 1$  different parts of the hypergraph, which are introduced if the hypergraph is constructed in this manner.

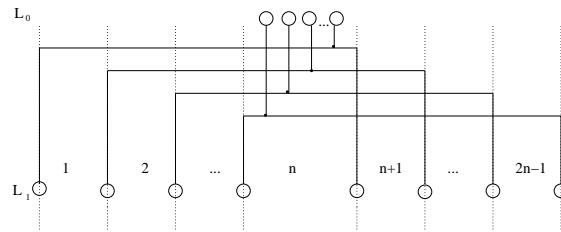


Figure 4: Transformation of the nodes.

Every hyperedge of this graph uses only one horizontal track. Two hyperedges have exactly two crossings with each other which leads to  $2 * ((n - 1) + (n - 2) + (n - 3) + \dots + (n - (n - 1)))$  different hyperedge crossings.

It is important to notice that the number of hyperedge crossings is independent of the assignment of hyperedges to horizontal tracks. This is due to the fact that swapping two hyperedges of adjacent tracks does not change the total number of hyperedge crossings, because of the symmetry of the arrangement. Therefore, without loss of generality, we use the simplified representation of the constructed hypergraph shown in Figure 5 for the rest of the proof.

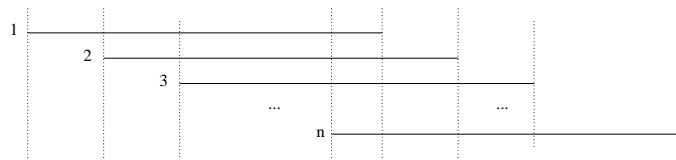


Figure 5: Transformation from a graph with  $n$  nodes without arcs.

Now, in the simplified representation of the hypergraph, we introduce a corresponding couple of nodes for every arc of the feedback arc set problem on layer  $L_1$ , which are represented by vertical lines. In general, an arc in the feedback arc set problem can point from a node with a smaller number to a node with a larger number, or vice versa.

**Case 1:** The arc points 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 from node 2 to node 4 is given.

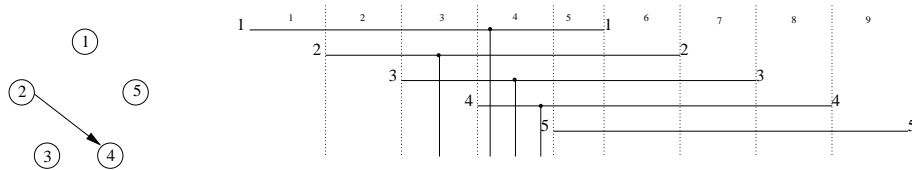


Figure 6: A graph with five nodes and one arc from node 2 to node 4 and its hypergraph.

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

**Case 2:** The arc points from 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 additional 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 additional vertical lines induce exactly  $c_{xy} + 1$  hyperedge crossings. In Figure 7, a transformation of a graph with five nodes and one arc pointing from node 4 to node 2 is given.

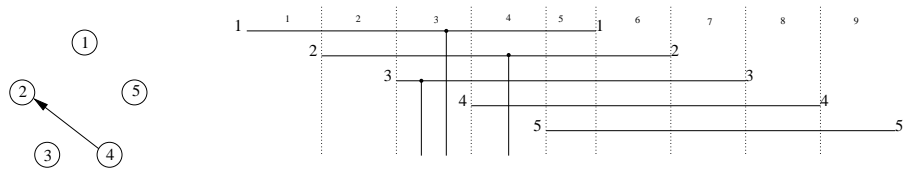


Figure 7: A graph with five nodes and one arc from node 4 to node 2 and its hypergraph.

Let  $C$  be  $\sum_{(x,y) \in A} c_{xy} + 2 * ((n - 1) + (n - 2) + (n - 3) + \dots + (n - (n - 1)))$ .  $C$  is a lower bound to the hyperedge crossing minimization problem because hyperedge  $h_i$  has to be placed before or after hyperedge  $h_j$  for all  $i \neq j$  and  $i, j \in 1, 2, \dots, n$ . 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. As there are at most  $k$  arcs in  $A'$ , the hypergraph can have at most  $\tilde{C} = C + k$  hyperedge crossings.

Otherwise, given an ordering  $\pi_h$  of the hyperedges such that the hypergraph has at most  $\tilde{C}$  hyperedge crossings, it is shown to imply a feedback arc set  $A'$  with  $|A'| \leq k$ . The ordering of the hyperedges  $\pi_h$  is mapped to the ordering of the nodes  $\pi_G$  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  $\pi_h$  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.  $\square$

Please notice that an optimal orthogonal embedding of all edges of a graph  $G = (V_0, V_1, E)$  (*GCM*) in this framework has the same number of crossings than the straight line embedding of the graph. It is also the number of crossings that occur if (without loss of generality) all nodes of layer  $V_0$  are placed left to all nodes of layer  $V_1$  and the edges (ordered from the first node in  $V_0$  to the last node in  $V_0$  and if more than one edge is incident to a node the edges are ordered with respect to the ordering given by  $V_1$ ) are assigned to the lowest up to the topmost track of the embedding.

## 4 Heuristic Methods

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

### 4.1 Greedy Assign

The greedy hyperedge assignment heuristic method iteratively assigns hyperedges starting at the top most track. First 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, fixing 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 =  $\sum_{j \in \text{nets}} c_{ij}$ 
      if(cost < mincost) {
        best = i
        mincost = cost
      }
      nets = nets \ best
    }
    assign best to track k
  }
}

```

## 4.2 Sifting

The sifting [24] algorithm was originally applied for minimizing the number of nodes in *Binary Decision Diagrams (BDDs)* [5], frequently used in logic synthesis applications and formal verification of logic circuits. Given all the crossing numbers, it is easy to apply the sifting algorithm to minimize the number of crossings. To get a first embedding the greedy assign method is applied. Then the algorithm chooses one net and moves it in the lowest track by repeatedly swapping it with its neighbor. Next, it is moved up to the topmost track. Finally, it is moved to its locally optimal position. Then the algorithm 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

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

In this case, sifting one net takes  $O(n)$  time, and sifting all nets summarizes to  $O(n^2)$  time.

The algorithm can be improved by an intelligent strategy to select the next node to be sifted. In the field of circuit visualization it seems to make sense to sift nets which are connected to many nodes before the nets which are only connected to a few nodes. It is also possible to use a randomly selected net. To obtain high quality results the algorithm repeats the sifting procedure for all nodes until it has reached a local optimum.

There are two well known methods to speed up the implementation:

- Moving the net first to the top most or the deepest track, which ever one requires the fewest swappings. This idea was first introduced by [24].
- Using upper and lower bounds to prune the search space, as introduced in [9].

Pseudo code for the algorithm is given below:

```

cmsifting(crossing_matrix cm, hyperedges nets){
  crossings = compute crossings of start permut.
  for i=1 to #nets{
    pos = actual track of net i
    for j = pos-1 down to 1 {
      k = net in track j
      swap net in track j with net in track j+1
      crossings = crossings -  $c_{ik}$  +  $c_{ki}$ 
    }
    best = crossings
    for j = 1 to #nets-1{
      k = net in track j+1
      swap net in track j with net in track j+1
      crossings = crossings -  $c_{ki}$  +  $c_{ik}$ 
      if(crossings < best){
        best = crossings
        bestpos = j+1
      }
    }
    for j = #nets-1 down to bestpos {
      swap net in track j with net in track j+1
    }
    crossings = best
  }
  return best
}

```

### 4.3 Reordering

Now we describe how the algorithms from the placement and routing phase are combined to further reduce the number of hyperedge crossings with respect to the hypergraph structure. Since finding exact solutions for both steps together implies to traverse a huge search space, two fast heuristic methods are used by turns to compute the final solution.

The new approach extends the well known greedy switch heuristic method for graphs to the hypergraph structure. It reorders iteratively two nodes in one layer and computes the orthogonal embedding of the hyperedges until a local optimum is reached. This new approach combines the placement and routing process with respect to the hypergraph structure in contrast to many known heuristic methods. The algorithm then repeats the reordering procedure until a local optimum is reached. The experimental results show that the runtime for all instances which easily fit on one screen are short which is crucial for an interactive circuit visualization tool.

Pseudo code for the reordering procedure is given below:

```

reordering(hypergraph hg) {
  for each layer {
    for each node in actual layer {
      before = compute crossings using sifting
      swap actual node with its right neighbor
      after = compute crossings using sifting
      if (before ≤ after) {
        swap actual node with its right
        neighbor
      }
    }
  }
}

```

Figure 8 shows the result after the crossing minimization step and the orthogonal embedding of the example circuit in Figure 1.

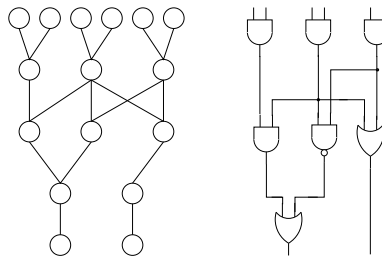


Figure 8: The graph after the crossing minimization step using the reordering and the orthogonal embedding of the circuit.

## 5 Experimental results

The algorithm has been implemented in *C*. All experimental results are based on examples which were taken from the sequential benchmark circuits in [4, 23]. The experiments were carried out on a 2 GHz personal computer with 1 GB main memory running linux OS. All running times are given in CPU seconds.

We utilized the barycenter heuristic method [28] combined with the greedy switch method to obtain a fast embedding of the nodes. Like observed in [19, 15], the barycenter method combined with the greedy switch heuristic computes high quality results in a short period of time compared to many other well known methods. The algorithm then applies the sifting procedure to assign each hyperedge to one track in the corresponding channel. This is done to get the orthogonal layout of each channel with respect to the number of crossings. Then the reordering algorithm improves the result by switching nodes within the same level and recomputing a orthogonal embedding, if the number of hyperedge crossings does not increase.

In Table 1 the results are summarized. In the second column we present the number of straight line crossings for each graph using the barycenter implementation combined with the greedy switch methods. In the third column the number of hyperedge crossings using sifting are given. The final result in terms of hyperedge crossings after applying the sifting and reordering procedure can be found in the fourth column. Compared to an assignment of the hyperedges over the tracks without using the reordering heuristic method the new method reduced the number of crossings by 25 percent on average. The running times of the algorithm on each circuit are shown in the last column. This demonstrates that in the VLSI CAD scenario, using the sifting heuristic together with the reordering method to compute the final orthogonal embedding is very effective.

Table 1: Benchmark results

circuit	edge crossings		h-edge crossings		time/s
	strl	sifting	siftplace		
add6	170	136	105		0.44
alu1	116	89	55		0.02
alu2	415	352	249		0.39
alu3	572	465	345		0.62
adr4	147	102	75		0.06
co14	74	65	54		0.09
dk17	367	274	203		0.2
dk27	77	64	38		0.03
dk48	427	378	305		0.39
mish	60	60	49		2.7
radd	59	48	38		0.03
rd53	253	172	141		0.04
s208	291	284	175		1.05
s298	904	550	434		2.16
s382	745	468	367		2.21
s386	1793	1254	853		1.89
s400	838	562	427		2.45
vg2	184	154	126		0.55
x1dn	195	175	136		0.76
x9dn	228	210	156		0.76
z4	117	91	60		0.07
Z9sym	3901	2320	1788		16
$\Sigma$	11933	8273	6179		32.91

In Table 2 it is shown that fewer edge crossings in the graph embedding lead in almost all cases to less crossings in the orthogonal drawing if only the sifting algorithm is applied. To obtain less crossings in the graph model we postprocessed the results computed with the averaging heuristic method with the windows optimization procedure [15]. The results are provided in the second and the third column, respectively. The windows optimization method uses a local optimization technique where subsets of nodes and edges are processed exactly. In contrast to most other heuristic methods more than two layers are

Table 2: Benchmark results

circuit	edge crossings		h-edge crossings		time/s	
	strl	winopt	sifting	splace	winopt	splace
add6	170	156	144	112	15	0.32
alu1	116	108	88	60	3	0.02
alu2	415	391	323	243	26	0.6
alu3	572	487	391	331	22	0.43
adr4	147	141	101	74	5	0.1
co14	74	69	65	52	6	0.13
dk17	367	308	249	188	17	0.22
dk27	77	71	57	45	2	0.02
dk48	427	397	341	290	32	0.29
mish	60	60	60	49	5	2.7
radd	59	52	43	37	2	0.04
rd53	253	218	152	126	14	0.09
s208	291	269	266	162	36	1.82
s298	904	854	515	428	27	2.38
s382	745	651	425	357	65	2.31
s386	1793	1636	1122	904	102	2.27
s400	838	725	512	400	59	3.63
vg2	184	168	151	131	9	0.58
x1dn	195	173	153	134	21	0.32
x9dn	228	206	185	158	12	0.27
z4	117	101	87	66	9	0.05
Z9sym	3901	3528	2160	1802	188	10.8
$\Sigma$	11933	10769	7590	6149	677	29.3

considered simultaneously. It turned out that using the windows optimization technique reduces the number of hyperedge crossings in the final orthogonal representation on average by eight percent. However, using the sifting and reordering heuristic method together with windows optimization instead of only sifting and reordering does not lead to significant improvements in terms of hyperedge crossings. This is because the final result does not depend as much on the given ordering of the nodes since the reordering heuristic improves the ordering of the nodes during the crossing minimization. In the fourth and in the fifth column the number of hyperedge crossings after applying the sifting and the sifting combined with the reordering algorithm can be found. The runtime for the windows optimization technique is given in the sixth column. The runtime for the sifting combined with the reordering algorithm are published in the last column. In the last rows of Table 1 and Table 2 we give the total number of crossings for each column.

## 6 Conclusions

We considered new methods that improve the visual clarity of circuit schematics. Between two consecutive layers any hyperedge is assigned to only one track and then the resulting number of hyperedge crossings is considered as optimization criterion.

At first, a polynomial transformation from the feedback arc set problem into the hypergraph crossing reduction problem to prove that the problem is NP-complete is provided.

Then the placement and the routing phase are merged by using two fast heuristic methods iteratively. Experiments have shown that the algorithm reduces the number of hyperedge crossings significantly compared to a layout method which does not reorder the nodes dynamically within a layer. Negligible runtime of our algorithm allows the visualization of large circuits in an interactive way.

## Acknowledgments

The authors wish to thank the referees for several useful suggestions.

## References

- [1] <http://www.onespin-solutions.com>.
- [2] D. Appenzeller and A. Kuehlmann. Formal verification of a PowerPC microprocessor. In *Int'l Conf. on Comp. Design*, pages 79–84, 1995.
- [3] P. Ashenden. *The Designer's Guide to VHDL*. Morgan Kaufman Publishers, 1995.
- [4] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. *Int'l Symp. Circ. and Systems*, pages 1929–1934, 1989.
- [5] R. Bryant. Graph - based algorithms for Boolean function manipulation. *TOC*, 35(8):677–691, 1986.
- [6] J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang. Symbolic model checking:  $10^{20}$  states and beyond. *Information and Computation*, 98(2):142–170, 1992.
- [7] 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.
- [8] R. Drechsler, W. Günther, L. Linhard, and G. Angst. Level assignment for displaying combinational logic. *EUROMICRO*, pages 148–151, 2001.
- [9] R. Drechsler, W. Günther, and F. Somenzi. Using lower bounds during dynamic BDD minimization. *IEEE Trans. on CAD*, 20(1):51–57, 2001.
- [10] P. Eades and D. Kelly. Heuristics for reducing crossings in 2-layered networks. *Ars Combin.*, 21(A):89–98, 1986.
- [11] P. Eades and N. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11:379–403, 1994.
- [12] 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*, pages 107–113, 2003.
- [13] T. Eschbach, W. Günther, and B. Becker. Orthogonal hypergraph routing for improved visibility. *Great Lakes Symposium on VLSI (GLSVLSI 2004), Boston, USA*, pages 385–388, 2004.
- [14] T. Eschbach, W. Günther, and B. Becker. Orthogonal circuit visualization improved by merging the placement and routing phases. *VLSI Design 2005 (VLSID'05), Calcutta*, pages 433–438, 2005.
- [15] T. Eschbach, W. Günther, R. Drechsler, and B. Becker. Crossing reduction by windows optimization. *Proceedings of the 10th International Symposium on Graph Drawing*, LNCS 2528:285–294, 2002.



- T. Eschbach et al., *Orth. Hypergraph Drawing*, JGAA, 10(2) 141–157 (2006)157
- [16] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NPCompleteness*. Freeman, 1979.
  - [17] M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 4:312–316, 1983.
  - [18] W. Günther, R. Schönfeld, B. Becker, and P. Molitor. K-layer straight-line crossing minimization by speeding up sifting. *Proceedings of the 8th International Symposium on Graph Drawing*, pages 253–258, 2000.
  - [19] 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.
  - [20] R. Karp. *Reducibility among combinatorial problems, in Complexity of Computer Computations (R.E. Miller and J.M. Thatcher, eds.)*. Plenum Press, 1972.
  - [21] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. Teubner, Wiley, 1990.
  - [22] C. Matuszewski, R. Schönfeld, and P. Molitor. Using sifting for k-layer straightline crossing minimization. *Proceedings of the 7th International Symposium on Graph Drawing*, LNCS 1731:217–224, 1999.
  - [23] K. McElvain. Benchmark set: Version 4.0. *International Workshop on Logic Synthesis*, 1993.
  - [24] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. *ICCAD*, pages 42–47, 1993.
  - [25] G. Sander. A fast heuristic for hierarchical manhattan layout. *Graph Drawing, Proc. Symposium on Graph Drawing, GD’95*, LNCS 1027:447–458, 1996.
  - [26] G. Sander. Layout of directed hypergraphs with orthogonal hyperedges. *11th Int’l Symposium on Graph Drawing 2003*, LNCS 2912:381–386, 2004.
  - [27] N. Sherwani. *Algorithms for VLSI Physical Design Automation*. Kluwer Academic Publishers, Norwell, Massachusetts, second edition, 1995.
  - [28] 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.