CONNECTIVITY BIASED CHANNEL CONSTRUCTION AND ORDERING FOR BUILDING-BLOCK LAYOUT

H. Cai

Delft University of Technology
Department of Electrical Engineering
Delft, The Netherlands

ABSTRACT

A number of techniques are presented for the construction and ordering of routing channels for building-block layout. First, before the routing channels are defined the placement is modified such that proper routing space is assigned between the circuit blocks. Second, a channel graph is constructed on which the global routing will be performed. Finally, after the global routing a feasible routing order is assigned to the channels. In contrast to other works, the algorithms use both the geometrical data (the placement) and the topological data (the connectivity) to decide which channel structure should be chosen from the feasible set.

1. Introduction

Building block based VLSI layout design is generally divided into three subproblems: placement, global routing and detailed routing. In channel based routing systems the formation of the channels establishes the interfaces between these three phases. During the past few years, many efficient algorithms were published to handle these three problems. However, not much attention has been paid on the interactions between these problems and interfaces between them. As handling the routing area correctly can have great impact on the quality of the final results, the routing-channel formation problem is studied in this paper.

In most channel based routing systems a channel graph is constructed which captures the adjacencies of the routing regions. On this graph the global routing is carried out to find a topological path for each net. Prior to the detailed channel routing a channel structure must be defined with a feasible routing order. With a feasible routing order the channels can then be physically routed at the detailed routing phase, one at a time without cyclic channel precedence constraints. Figure 1 gives an overview of the subsequent steps in our layout system.

In this paper we present algorithms to perform the steps (2), (3) and (5).

In many systems a channel graph is derived from a given placement of the blocks [Kimu83, Dai85]. The channel graph obtained in this way is strongly dependent on the absolute positions of the blocks. As most placement algorithms do not have accurate routing area estimation procedures, the space allocated for the connection wires prior to the routing itself usually differs from the actual routing space needed. This has great influence on the global routing accuracy, and what is most important, on the decisions in constructing the channel graph. We propose an algorithm to adjust the routing space accurately prior to the channel graph construction step. The details of the algorithm are described in section 3.

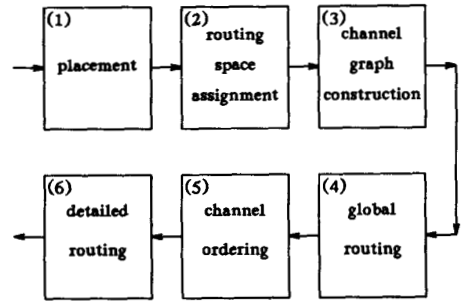

Figure 1. Overview of the layout system.

Given a placement of the blocks, many different channel structures can usually be defined. For example the placement of five blocks in Fig. 2.a has four feasible channel structures (see Fig. 2.b) which will usually produce different routing results.

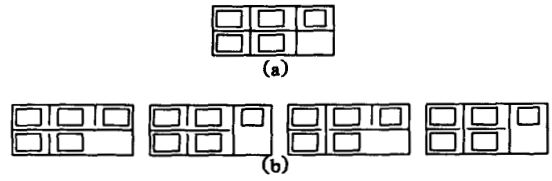

Figure 2. Feasible channel structures.

In this paper we consider the problem of choosing the best channel structure from the feasible set. This is done in two steps, step (3) and (5) in Fig. 1. First, a channel graph is constructed. The channel graph obtained from the placement usually contains empty rooms, see Fig 2.a, which must be eliminated before routing. An algorithm is presented which eliminates the empty rooms from the channel graph by deleting selected edges and vertices.
In contrast to most routing systems which define a channel order before the global routing phase [Kimu83, Laut85], we propose an algorithm to define the channels and the channel routing order after the global routing phase. The main advantage of this approach is that global routing results can be taken as additional information to make more intelligent decisions in determining the channel order. Details of these two algorithms are given in section

4 and 5.

2. The layout model and definitions

We assume that circuit blocks are rectangular in shape with terminals located at the boundary of the blocks. The routing regions are represented by a *channel graph*, see for example Fig. 3. The channel graph is defined in a similar way to the floor plan graph defined in [Dai85]. Vertices represent the channel intersections and there is an edge between two vertices if the corresponding channel intersections are adjacent to each other. Each edge corresponds to a channel segment. A routing channel is a straight line formed by a set of consecutive channel segments between two channels in the perpendicular direction.

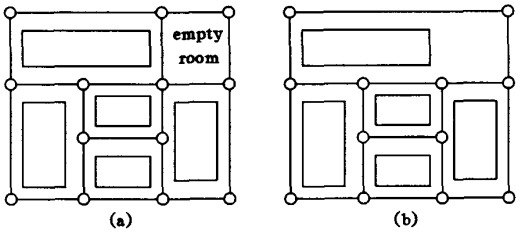

Figure 3. (a) a channel graph, (b) a feasible channel graph.

The following list gives some relevant definitions.

- *Room:* a region bounded by channel segments but containing no channel segments is called a room.
- *Empty room:* a channel graph may sometimes contain empty rooms which are rooms that do not contain a block, see Fig. 3.a.
- *Channel structure:* a channel structure is a set of channels derived from an empty room free channel graph which dissect the placement.
- *Routing order:* a routing order is an ordering of the channels in which a channel is assigned an order which is higher than that of all its intersecting neighboring channels.
- *Feasible channel structure:* a feasible channel structure is a channel structure for which a routing order exists.
- *Feasible channel graph:* a channel graph from which a feasible channel structure can be derived is called a feasible channel graph. Figure 3.b shows an example of a feasible channel graph.
- *Slicing placement:* a placement for which a feasible channel structure exists is called a slicing placement.

3. The routing space assignment algorithm

As the absolute placement of the blocks strongly influences the channel structure derived from it, a realistic placement which is a placement that almost matches the final layout is highly desirable. The routing space assignment algorithm repositions the blocks before the routing phase such that this match is optimized.

Inspired by the constraint graph method in the layout compaction field [Cho85] we construct two *channel position graphs*, one in each direction, see also [Laut79, Prea78], to represent the adjacencies of the routing regions and the blocks. In the vertical-channel position graph each vertex represents the top side edge or the bottom side edge of a block and an edge represents a dimension, either a block or a channel segment between two blocks. Two additional vertices, s and d, are added to the graph as the source and destination vertices. The horizontal-channel position graph is defined similarly. Figure 4 gives an example of such a pair of directed graphs.

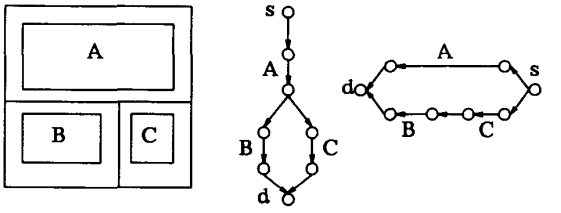

Figure 4. Channel position graphs.

The algorithm repositions the blocks such that proper routing space is provided. The blocks are moved in one direction at a time. Let us consider the vertical direction (the horizontal direction is processed in a similar way). First, a channel graph is constructed using the algorithm to be described in the following section. However, possible empty rooms in the graph are not removed at this stage. They are temporarily filled with dummy blocks of size zero. The channel position graph is derived from the channel graph and the block placement. In the next step, all paths from the source vertex s to the destination vertex d in the channel position graph are searched and sorted in order of decreasing path length. The length of a path is the sum of the length of its edges which represent the height of the blocks and the width of the channel segments. The height of the blocks is known. However, the width of the channel segments needs to be calculated. For this purpose the global routing routine is invoked to find a shortest connection for each net. After this operation the number of wires in each channel segment multiplied by a constant is used as an approximation of the width of the channel segments.

Given the longest path from s to d, L_{sd}, and the longest path from s to d, l_{sd}, but without taking into account the width of the channel segments, an estimate of the chip height H is calculated using the following formula:

$$H = l_{sd} + C \times (L_{sd} - l_{sd})$$

where C is a packing constant between 0 and 1 (default 0.95). Note that if C = 1 enough space is allocated for all channel segments; if C = 0 no routing space is allocated, the algorithm becomes a block-packing procedure.

If the blocks are repositioned such that the estimated chip height is realized, the paths longer than H will have wiring overflows in some of their channel segments and the paths shorter than H will contain channel segments with dead space. Figure 5 shows an example of a list of ordered paths from s to d and an estimated chip height. The motivation for creating some overflows artificially ($C<1$) in some channel segments is based on the expectation that the global router will fill the dead space with the overflowed wires using a congestion-biased cost function. This will result in a more uniform wiring distribution and smaller chip area.

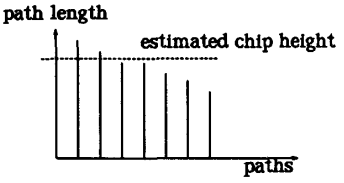

Figure 5. List of ordered paths.

The total overflow or dead space on a path is divided among its channel segments. To each channel segment a fraction of the total overflow or dead space is assigned, with a percentage proportional to its width. Consequently some channel segments will become either narrower or wider than the minimum required width. As a channel segment can be a part of more than one path, the paths are processed in the order of decreasing path length. This ordering implies that the more critical paths are processed first. After a channel segment is assigned a width it is "locked" against further change. If, in a path, some channel segments are locked, the remaining overflow or dead space on the path is assigned to the unlocked channel segments. Finally, using the channel segment widths just determined the blocks are moved to their longest distance positions from the destination vertex.

Compared to the case in which the blocks are moved directly to the longest path distance using the minimum channel segment widths, an advantage of this method is that the dead space is distributed among the channel segments evenly over the chip area instead of being concentrated in the upper and right chip boundaries.

The procedure may be iterated to achieve a more accurate placement. The horizontal and vertical directions are processed in an alternative order. However, too many iterations are expensive, because at each iteration a global routing is performed. Note that if the final placement is not a slicing placement, it is modified to a slicing placement. This is because our routing system accepts only slicing placements.

4. Construction of a feasible channel graph

4.1 Existing approaches

To construct a feasible channel graph from a slicing placement, generally, two approaches can be applied. The first one is the hierarchical approach [Laut85] in which the set of blocks is recursively partitioned by straight slice lines until each subset consists of only a single block. The set of slice lines together with the placement boundary lines form a feasible channel graph by construction. This set of lines are often derived during the automatic placement phase, for example using the min-cut algorithm [Laut79]. Further, if the slice lines are used as routing channels in the detailed routing phase, the inverse order of the partition lines is a valid channel-routing order. The second approach is the flat approach [Kimu83]. In this approach the blocks are treated all at the same time. Line segments are extented from the boundaries of the blocks until they hit a block. Then the number of lines is reduced by merging them into each other as much as possible. The intersections of the remaining horizontal and vertical lines form a channel graph containing all feasible channel graphs from which one must be chosen. Note that the flat approach can also be applied to nonslicing placements.

The hierarchical approach is more efficient than the flat approach, however not without a price. Since it treats the two subsets divided by a slice line independently, it looses the global view of the placement as a whole. Consequently, a feasible channel graph is chosen arbitrarily in this approach. The decision on the channel routing order is made earlier than necessary, that is long before the detailed routing. In our system, this decision is postponed until after the global routing. In contrast to the hierarchical approach, the channel graph constructed by the flat approach may contain empty rooms, which must be eliminated to obtain a feasible channel graph. Different ways of eliminating the empty rooms yield different feasible channel graphs. In [Kimu83] this problem is not mentioned. In [Dai85] a method is proposed to remove the empty rooms in channel graphs by deleting one of the edges adjacent to an empty room during the channel definition process. However, counter examples can be shown that when empty rooms are situated adjacent to each other, this method may fail to find a correct solution

4.2 Feasible channel graph construction algorithm

To construct a feasible channel graph for a slicing placement, we propose a method combining the flat and the hierarchical approaches. First, a channel graph is generated using the flat approach according to [Kimu83]. Note that the channel graph used in the routing space assignment algorithm is generated by the flat approach. Then the empty rooms are eliminated by recursive bisection of the placement along the channel segments. This means that at each recursion level the blocks and the channel segments are partitioned into two subsets such that each contains at least one block. This process is continued until each subset contains exactly one block. Simultaneously, a *partitioning tree* is constructed. At each node of the tree the blocks and the channel segments inside the corresponding subset are recorded.

An example of such a partitioning tree is shown in Fig. 6.

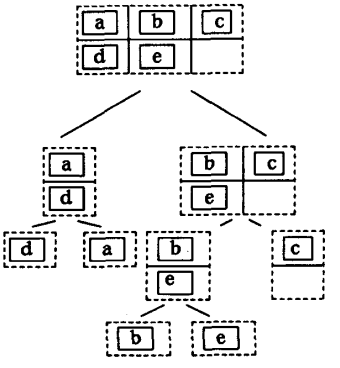

Figure 6. A partitioning tree.

The dashed lines represent the boundary of each subset. Note that the channel segments on the boundary of a subset are not stored at the corresponding node, only those inside the subset are stored. After the tree is constructed the remaining channel segments in all leaf nodes are removed. A feasible channel graph can then be easily

obtained by generating a vertex for each line intersection and proper edges between them.

The feasible channel graph construction algorithm based on the idea described is shown below:

Feasible channel graph construction algorithm

INPUT: a slicing placement

OUTPUT: a feasible channel graph.

METHOD:

/ derive initial channel graph with the flat approach */*
Step 1): Draw horizontal and vertical lines extending the edges of each block. If such a line intersects an edge of another block, then cut it at the interconnection.

Step 2): If, for any two line segments thus drawn, the interval of one is wholly covered by that of another and there is no block between them, then merge them.

Step 3): Determine the intersection points of the two set of lines; keep only the line segments between two intersections.

/ remove empty rooms with the hierarchical approach */*
Step 4): Initialize the root of the partitioning tree with all blocks and line segments.

Step 5): Partition (root).

Step 6): Generate the channel graph by assigning a vertex to each line intersection and edges between the adjacent vertices.

Procedure *Partition (S)*

if the number of blocks in S ≡ 1
{
Remove all remaining line segments in S.
}
else
{
Choose a slice line which partitions S [see 4.3].

Divide the blocks and line segments in S into S' and S''.
(the line segments on the slice line are not taken)

Partition (S'), Partition (S'').
}

4.3 Heuristic to choose a slice line

"*Choose a slice line which partitions S*" in the procedure "*Partition*" is crucial for the quality of the routing results. Generally, in the presence of empty rooms, different partitioning sequences result in different feasible channel graphs. For instance, for the placement in Fig. 7.a, two feasible channel graphs are possible, Fig. 7.b and Fig. 7.c. Intuitively, the horizontal slice line should be chosen in the example which results in the feasible channel graph in Fig. 7.b, because of the large wiring flow on that slice line (represented by the dashed lines); otherwise it will result in wiring detours, see Fig. 7.c.

The following heuristic has been resorted to choose a slice line. As shown in the example the choice should depend on the wiring flows (traffic) on the channel segments. Therefore, on the initial channel graph the global router is called to derive the number of wires on each channel segment. A slice line usually consists of several channel segments. To each possible slice line a weight is assigned which is equal to the sum of the number of wires on its channel segments that are adjacent to an empty room. Channel segments that are not adjacent to an empty room do not contribute to the weight of the slice line. Further, on a channel segment the wire segments which pass the channel segment get a higher weight than those which start or end in it, because passing wires imply that the channel segment is on the shortest path of those wires. Therefore deleting this channel segment would cause routing detours. The slice line with the highest weight is chosen.

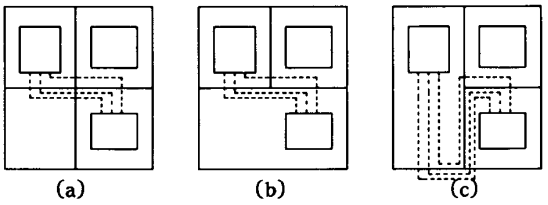

Figure 7. Feasible channel graphs.

5. The channel ordering algorithm

After the global routing, a channel order must be defined before proceeding to the detailed routing phase. Often the channel graph contains vertices with degree 4. This type of edge intersections is called a "+" type channel crossing. Other vertices form "T" type channel intersections. To define a channel order all "+" type channel crossings must be transformed into two "T" type channel intersections in one of the two ways. Hence, given a feasible channel graph more than one possible channel order may exist caused by the presence of "+" type channel crossings. However not all combinations of "+" type crossing transformations yield a feasible channel structure, because channels must be ordered such that the channel structure does not contain channel precedence cycles. In this section a channel ordering algorithm is proposed to derive a feasible channel structure.

The decision in which direction a "+" type crossing will be split depends on two criteria in this algorithm. First, the channels on the critical (longest) paths in the channel position graphs are kept as short as possible. In this way the possible negative influence between these channels and the channels not on the critical paths is avoided. We call this criterion the *critical path isolation* criterion. Second, the decision depends on the wiring flow (traffic) across the "+" crossing area. It is preferable to use the direction with the largest wiring flow for the main channel in order to minimize the number of wiring jogs and to avoid congestions in the crossing area. We call this criterion the *major road* criterion. An example of the wiring flow in a "+" type crossing area is shown in Fig. 8. In this example the channel segments "a" and "b" should obviously be kept in one channel (the major road).

We assign a weight to each direction in which a "+" type crossing is split indicating the preference of splitting the crossing in this direction. A "+" type channel crossing is said to be split in the horizontal direction if the two channels in this direction are separated by the main channel in the vertical direction. The same applies to the vertical direction. According to the criteria mentioned above, for

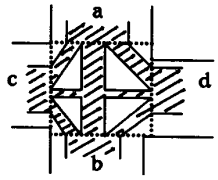

Figure 8. Wiring flow in + type channel crossing.

each "+" type crossing adjacent to a channel segment on a critical path, a weight 2 is assigned to the splitting in the direction perpendicular to the direction of the critical path (the critical path isolation criterion). For each "+" type crossing a weight 1 is assigned to the splitting in the smallest wiring flow direction (the major road criterion). To all other ways of crossing splitting a weight 0 is assinged. The best feasible channel structure is the one with the highest total weight of crossing splittings which is the target of the channel ordering algorithm.

The underlying idea of the channel ordering algorithm is explained using a simple example. Consider a channel graph containing 3 "+" type crossings (c1, c2 and c3) in Fig. 9.

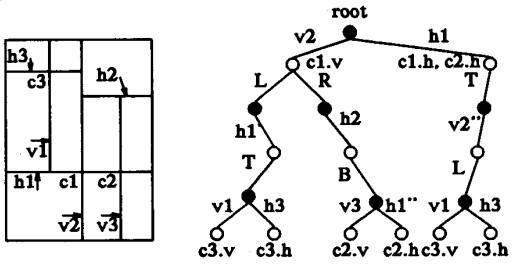

Figure 9. A crossing decision tree.

We construct a so-called *crossing decision tree (cd_tree)* representing all legal combinations of crossing splittings as follows: starting at the root level, at each level the channel graph is sliced in all possible ways: for example slice line v2 and h1 in the example at the root level. Two kinds of nodes are distinguished in the tree. Each slice line is represented by a white node in the tree. The two subsets partitioned by the slice line are represented by two black child nodes of the white node. For example slice line v2 will lead to a white node and the left and right subsets of v2 are represented by two black nodes.

At a white node all "+" type crossings split by the corresponding slice line are recorded. The way a crossing is split by a slice line is indicated at the white node by a direction suffix after the crossing name, for instance, c1.h indicates that a horizontal slice line is used, hence c1 is split in the vertical direction. To each white node a weight is assigned which is equal to the total weight of the crossing splittings by the corresponding slice line. The name at the side of an edge from a black node to a white node is the name of the slice line used; the character at the side of an edge from a white node to a black node indicates the side of a subset with respect to the parent slice line. Each black node is processed recursively according to the strategy described above. However, we can terminate the process in a branch if a subset does not contain "+" type crossings any more. To further reduce the size of the tree slice lines which do not split a "+" crossing are processed first and are assigned to separate levels in the tree. For example, slice line h2 is processed before slice line h1" which is the portion of h1 on the right side of v2.

Each sub-tree in the cd_tree that splits each "+" type channel crossing in the graph once and only once corresponds to a feasible channel order. Now the problem of choosing the best channel order is reduced to finding the sub-tree of the cd_tree with the highest total weight of the crossing splittings.

To find the sub-tree with the maximum weight, we developed an algorithm akin to finding the longest path in acyclic graphs. The algorithm traverses the tree in a postorder. We associate with each node v_i a weight $W(v_i)$. The weights are easily calculated by scanning the child nodes for all nodes. The weight of a black node is the maximum weight of its child nodes; and the weight of a white node is the sum of the weight of its child nodes and the weight on the white node itself assigned during the tree construction.

After the tree traversal, the resulting sub-tree is traced by starting at the root, recursively collecting the maximum-weight white child node of a black node and the black child nodes in the case of a white node. For our example in Fig. 9, with the following input crossing weights:

W(c1.v) = 1 W(c1.h) = 0
W(c2.v) = 0 W(c2.h) = 1
W(c3.v) = 1 W(c3.h) = 0

the resulting maximum-weight sub-tree is indicated by the dotted line shown in Fig. 10. The value on a node is the weight at that node. The optimal combination of the crossing splittings is found: c1.v, c2.h and c3.v with a total weight 3.

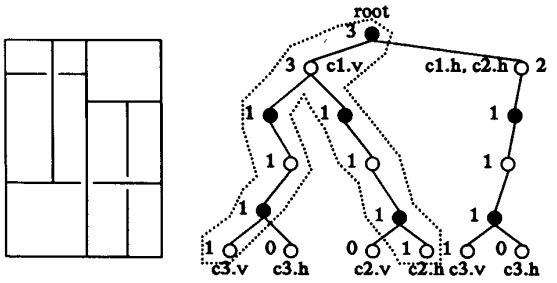

Figure 10. The maximum-weight sub-tree of the tree in Fig. 9.

The procedure form of the Channel-Ordering algorithm is shown below:

Channel-Ordering algorithm

INPUT: a feasible channel graph

OUTPUT: an ordered sequence of routing channels.

METHOD:
{
Assign the crossing splitting weights.

Build_cd_tree (channel graph).

Find the maximum-weight sub-tree.

```
Slice the graph according to the crossing splittings,
yielding the channels and the channel order.
}

Procedure Build_cd_tree (S)
{
If (there are "+" type crossings in S)
   {
   Make a black node for S.

   For (all possible slice lines that split S into two parts S' and S")
        {
        Make a white node and record the crossing weights
        split by the slice line.

        Build_cd_tree(S'), Build_cd_tree(S").
        }
   }
}
```

The size of the cd_tree which, in the worst case, is proportional to the sum of the number of slice lines at each slicing level, can become large, if there are many slice lines at each level and some "+" type crossings are hidden in very low levels. In that case the algorithm should be applied hierarchically. This can be realized by imposing a maximum size of the tree and applying the algorithm again in lower levels.

6. Implementation and experimental results

The algorithms were implemented in the C programming language under the UNIX operating system. They have been integrated into our placement and routing system [Cai85, Groe87]. The system has been successfully used in designing a number of large chips.

Experiments on two chip examples are reported in this section. These two chips, called Xerox and ami33, were obtained from the International Workshop of Placement and Routing in North Carolina this year as standard benchmark chips at the workshop. The essential data of the two chips is given in table 1.

chip	#blocks	#nets	#terminals
Xerox	10	203	698
ami33	33	123	522

TABLE 1. Test chip data.

To test the routing space assignment algorithm, a placement is generated for each of the chips. The blocks are placed as closely as possible to each other. The detailed routing results are shown in table 2 and 3.

chip	chip size	net length	#vias	CPU time(s)
Xerox	35124200	1117133	1560	17.4
ami33	3218360	173962	1045	24.4

TABLE 2. Routing result without using the routing space assignment algorithm.

chip	chip size	net length	#vias	CPU time(s)
Xerox	34305400	1039832	1429	21.5
ami33	2968690	160855	962	33.5

TABLE 3. Routing result of the same placements which are first modified by the routing space assignment algorithm.

The CPU time reported is that on a Apollo DN3000 workstation and is the time to perform the steps from (3) to (5) (table 2) or from (2) to (5) (table 3) in Fig. 1. From the tables we observe an average improvement around 7% in the chip size, the total net length and the number of vias at an average increase of about 30% in CPU time. Note that the increase in CPU time with respect to the total routing time (including the detailed routing time) is very small.

To see the performance of the channel graph construction and channel ordering algorithms a placement for each chip is also routed twice, once using manually entered channels with random choices of slice lines and once using the two algorithms to derive a channel structure, see table 4 and 5.

chip	chip size	net length	#vias	CPU time(s)
Xerox	34566800	1032937	1470	16.4
ami33	3207140	159175	1021	20.7

TABLE 4. Routing result using manually entered channels with random choices of slice lines.

chip	chip size	net length	#vias	CPU time(s)
Xerox	34550400	1039595	1425	17.1
ami33	2987580	158293	983	23.4

TABLE 5. Routing result using a channel structure derived from the two algorithms.

From the tables we observe an average improvement around 2.5% in the chip size, the total net length and the number of vias at an average increase of about 9% in CPU time.

7. Conclusions

Although the algorithms presented are heuristic in nature, they have been proved to perform well in a practical environment. Beside geometrical information, wiring information is also used in the channel construction and ordering process. Decisions are postponed as long as possible to avoid making wrong decisions at early stages. These techniques provide a smooth transition from the placement to the global routing to the detailed routing steps and some feedback from a later step to an early step.

References

Cai85. H. Cai, H. Th. Verheyen, R. Nouta, and P. Dewilde, "An Automatic Routing System for General Cell VLSI Circuits," *Proc. Custom Integrated Circuits Conference*, pp. 68-71, Portland, USA (May 1985).

Cho85. Y. E. Cho, "A Subjective Review of Compaction," *Proc. 22nd Design Automation Conference*, pp. 396-404 (1985).

Dai85. W. M. Dai, T. Asano, and E. S. Kuh, "Routing Region Definition and Ordering Scheme for Building-Block Layout," *IEEE Transactions on Computer-Aided Design* **CAD-4**(3) pp. 189-197 (July 1985).

Groe87. P. Groeneveld, H. Cai, and P. Dewilde, "A Contour-Based Variable-Width Gridless Channel Router," *Proc. ICCAD*, pp. 374-377, Santa Clara, USA (1987).

Kimu83. S. Kimura, N. Kubo, T. Chiba, and I. Nishioka, "An Automatic Routing Scheme for General Cell LSI," *IEEE Transactions on Computer-Aided Design* **CAD-2**(4) pp. 285-292 (October 1983).

Laut85. U. Lauther, "Channel Routing in a General Cell Environment," *Proc. VLSI*, pp. 389-398 (1985).

Laut79. U. Lauther, "A min-cut placement algorithm for general cell assemblies based on a graph representation," *Proc. 16th Design Automation Conference*, pp. 1-10 (1979).

Prea78. B. T. Preas and C. W. Gwyn, "Methods For Hierarchical Automatic Layout," *Proc. 15th Design Automation Conference*, pp. 206-212 (1978).