INTEGRATION, the VLSI journal 14 (1992) 49–65
Elsevier

A new algorithm for standard cell global routing

Jason Cong

Department of Computer Science, University of California at Los Angeles, Los Angeles CA 90024, USA

Bryan Preas

Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304, USA

Received 5 November 1990
Revised 11 May 1992

Abstract. In this paper, we present a new algorithm for standard cell global routing. The algorithm considers all of the interconnection nets simultaneously; this produces superior results since information about all of the nets is available throughout the global routing process. We formulate the global routing problem as one of finding the optimal spanning forest on a graph that contains all of the interconnection information. The results of an important theorems allow us to prune many non-optimal connections before the global routing process begins. This approach successfully solves the net ordering and congestion prediction problems which other approaches suffer. The new algorithm was implemented as part of the DATools system at Xerox PARC. The benchmarks from the Physical Design Workshop are used as part of the comparison suite. The new algorithm achieves up to 11% area reduction compared to the previous global routing package used in the DATools system and obtains up to 17% reduction in the total channel density compared to the Timberwolf 4.2 package.

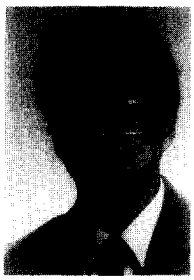
Keywords. Global routing; standard cell design; spanning forests.

1. Introduction

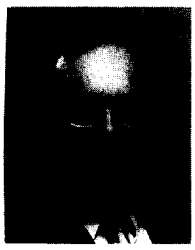
The standard cell design style is widely used in the design of VLSI circuits; the cells (either obtained from a library or constructed by a cell generator) are arranged in horizontal rows. The aim of a standard cell design system is to generate a correct physical design for a circuit from its logic design with the best

utilization of chip area. Due to the inherent complexity of physical design, the process is usually divided into three steps: placement, global routing and channel routing. During placement, we determine the row and the position within the row for each cell in the logic design. Next, during global routing, we determine the connection pattern, or topology, for each net. Global routing introduces *feedthrough* cells making connections across the cell rows, selects the pins (from the pins that are internally connected within the cells) to be connected, and determines the routing channels that the net segments belong to. Then, each channel is routed individually by a channel router which assigns specific layers and tracks for wires to implement the connection patterns determined by global routing. This paper studies the global routing problem in standard cell designs.

The global router in the Highland system [1] was used as the benchmark for standard cell global routing at the Physical Design Workshop on Placement and Floorplanning [2]. It builds aluminimum spanning tree for each net. The cost for each net edge is a function of channel density, feedthrough availability and wire length. An subsequent optimization step tries to improve the solution. Supowit [3] gives an odd-even heuristic algorithm for standard cell global routing. It produces a solution within a factor of 1.5 of the optimal solution (in terms of total channel density). However, Supowit's result applies only to problems in



Jason Cong received his B.S. degree in computer science from the Peking University in 1985. He received his M.S. degree and Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign in 1987 and 1990, respectively. Currently, he is an assistant professor in the Computer Science Department of University of California, Los Angeles. From 1986 to 1990, he was a research assistant in the Computer Science Department of the University of Illinois. He worked at the Xerox Palo Alto Research Center in the summer of 1987. He worked at the National Semiconductor Corporation in the summer of 1988. His research interests include computer-aided design of VLSI circuits, faulttolerant design of VLSI systems, and design and analysis of efficient combinatorial and geometric algorithms. He received the Best Graduate Award from the Peking University in 1985. He was awarded a DEC Computer Science Fellowship in 1988. He received the Ross J. Martin Award for excellence in research from the University of Illinois at Urbana-Champaign in 1989. He received the National Science Foundation Engineering Research Initiation Award in 1991.



Bryan T. Preas obtained the B.S. degree from Texas A&M University in 1968, the M.S. degree from Carnegie-Mellon University in 1969, and the Ph.D. degree from Stanford University in 1979. He was with Bell Telephone Laboratories, Whippany, NJ from 1968 to 1973 where the worked on hardware and software design for large, multiprocessor computers. From 1973 to 1981 he was with Sandia National Laboratories, Albuquerque, NM with the CAD development group. He was Vice President of Research and Development at VR Information Systems in Austin, TX from 1981 to 1983 where he was responsible for development of CAD software. In 1983, he joined the Xerox Palo Alto Research Center as Area Manger of Design and Architecture and Principal Scientist. Dr. Preas was awarded the Humboldt Senior Scientist prize and spent 1990 at the University of Paderborn, Germany. Dr. Preas has published numerous technical papers and is active in several professional organizations. He serves on the program committees of several conferences and is Associate Editor of the IEEE transactions on CAD. He is currently Program Chair of the Design Automation Conference.

which all the nets are two-pin linear nets. Another global router developed for standard cell designs is part of the Timberwolf package [SeSa86]. It first connects each net by a minimum spanning tree based on wire length. Then it employs an iteration algorithm (simulated annealing with $t = 0$) to improve the assignment of net segments to channels. The previous standard cell global router [4] used in the DATools system at Xerox PARC [5] generates the minimum number of feedthroughs. Feedthroughs are added only to connect the nets. Feedthroughs are then placed, along with the other cells, as part of a row-based placement improvement step. More recent work on standard cell global routing includes the work by Mowchenko and Ma [6], which generalized the left edge algorithm for channel routing to global routing, the work by Lee and Sechen [7] which generated Steiner trees from minimum spanning trees, a parallel router by Rose [8], which enumerated all possible two-bend nets, a parallel router by Brouwer and Banerjee [9], which is based on Burstein and Pelavin's hierarchical approach, and a global router by Meixner and Lauther [10] based on network flow computation. In [11] and [12], a similar problem was addressed for gate array designs, where the objective was to minimize the maximum channel density.

A careful study shows that these approaches have one or more of the following shortcomings:

(1) The final solution produced is sensitive to the order in which the nets are considered because the nets are connected one by one. However, little is known about what is a good net order. (In [6], they avoided net ordering problems by carrying out the routing on a channel by channel basis. But still, it is difficult to choose a good channel order.) [1].

(2) These global routers are not capable of predicting congested area in channels when they add net segments. This is especially true in the early stage when most net segments have not been included.

(3) The net connection patterns that can be produced are restricted by the algorithms. Only a few predetermined topologies are allowed.

In this paper, we present a new algorithm for standard cell global routing which successfully overcomes the shortcomings mentioned above. This algorithm processes all the nets in parallel, so the results are independent of the order in which the nets are considered. Furthermore, better results are produced since information about all of the nets is available throughout the global routing process. We introduce the net connection graph and formulate the problem as finding an optimal spanning forest of the net connection graph. We prove a theorem which allows us to simplify the net connection graph by pruning a large number of non-optimal connections. This makes it computationally feasible to consider the optimal connections in all the channels at the same time. Thus, our algorithm can predict very accurately the densest areas in each channel and, therefore, distribute density evenly over all channels to minimize the *total channel density*. The new algorithm was implemented as part of the DATools

[1] Parallel global routers usually avoid the net ordering problem.

system at Xerox PARC. Benchmarks from the Physical Design Workshop are used as part of the comparison suite. The new algorithm achieves up to 11% reduction in area compared to the previous global routing package used in the DATools system and obtains up to 17% reduction in the total channel density compared to the Timberwolf 4.2 package. In no case does the new algorithm do worse than its competitors.

The remainder of this paper describes the algorithm. Section 2 defines the problem formulation. The two-stage algorithm is discussed in Section 3. The comparative results are presented in Section 4. An extended abstract of this paper was presented in ICCAD'88 [13].

2. Formulation of the problem

The goal of global routing is to determine the connection pattern for each net and achieve the best utilization of chip area. The connection pattern is defined by the positions for feedthroughs, the pins to be connected, and the channels in which the net segments that connect the pins lie. *Chip area* is equal to the product of the width of the chip and the height of the chip, where the *width of the chip* is the maximum length (including any feedthroughs) of any row of the chip, and the *height of the chip* is the sum of the heights of all cell rows and the total channel density times the line-to-line spacing.

We define the *net connection graph* to be an undirected graph $G = (V, E)$, where V is the set of pins currently in the design. An edge (p_i, p_j) is in E if the two pins p_i and p_j are in the same net. We also call (p_i, p_j) a *net segment* if p_i and p_j are in the same channel. Clearly, each net in the net connection graph is a *connected component* and is represented by a complete graph on the pins of the net. Note that V may grow as we perform the global routing because new pins are introduced when feedthroughs are added. A *global routing solution* is a spanning forest of the net connection graph. A spanning forest which yields the minimum chip area is called an *optimal spanning forest*.

There are two problems involved in standard cell global routing. The first problem is to determine whether and where to add feedthroughs. Generally speaking, feedthroughs have two functions. One function is to complete the

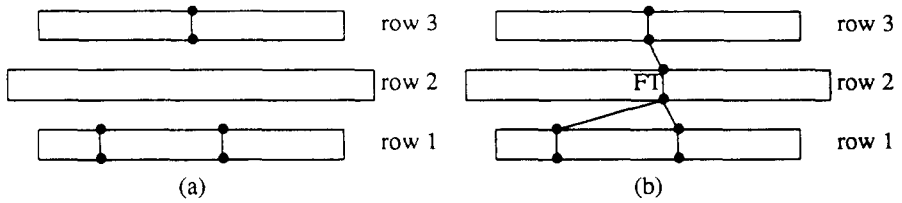


Fig. 1. A feedthrough allows a net to cross a cell row. The feedthrough in row 2 is required to complete the connection.

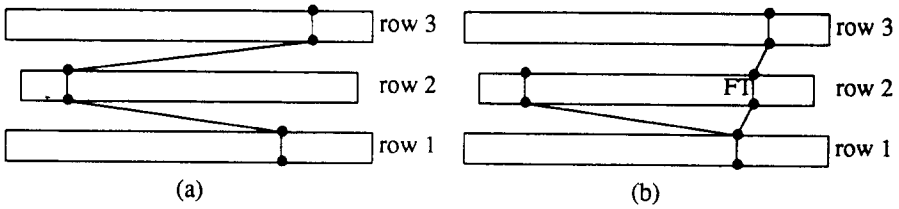


Fig. 2. Feedthroughs can also be used to reduce the total channel density.

connections among pins that make up the nets. For the example shown in Fig. 1(a), we have to insert a feedthrough in row 2 to complete the connections for the net as shown in Fig. 1(b). The other function of feedthroughs is to reduce the total channel density. Consider the net shown in Fig. 2(a). Although we can complete the connection without adding any feedthroughs, by adding a feedthrough in row 2 we save a long wire in channel 2. This may reduce the total channel density. The second problem in standard cell global routing is to determine the net segments to complete the connection of the nets after feedthroughs have been added. Figure 3 shows three different choices of net segments to connect a net. At this stage, since the width of the chip is fixed, the problem is to build a spanning forest of the net connection graph to minimize the total channel density.

In some standard cell families, many cells have build-in feedthroughs. In this case, the feedthrough insertion problem is eliminated or simplified. In [3] and [6], it was assumed that all the feedthroughs have been added and only the problem of determining the net segments was studied. In [4] and [14], simple methods were used to determine feedthrough locations for completing the connections. Their algorithms concentrated on the problem of determining the net segments. In our global router, we also use a rather straightforward method to compute the feedthroughs first. Then, we focus on the optimal selection of net segments.

The standard cell global routing problem is computationally difficult; we can show the problem of finding an optimal spanning forest is NP-hard. In fact, the problem of determining net segments itself is already NP-hard even for a small number of cell rows (assuming that no feedthroughs need to be added). To be more precise, we state the following theorem based on a result in [15]:

Theorem 1. *Given a standard cell placement in which no feedthroughs are needed to complete the net connections, the problem of choosing net segments to minimize total channel density is* NP*-hard if there are five or more cell rows in the design.*

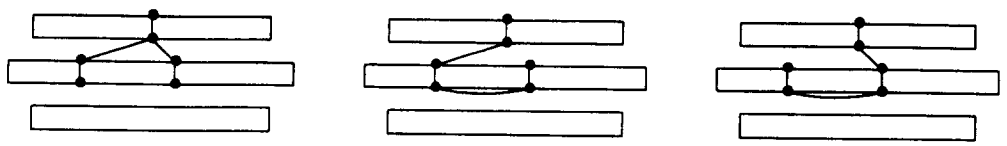


Fig. 3. For the net shown, three different choices of net segments.

Proof. First, let us look at a simplified net segment selection problem. Suppose that each net has two logical pins. Moreover, the two logical pins in each net are in the cells of the same row. Such a net is called a *two-pin linear net* [3,16]. Since each logical pin has two physical pins, there are two possible ways to connect each linear net, one way to use the channel below the net and the other to use the channel above the net (see Fig. 4). We want to connect each two-pin linear net so that the total channel density is minimum. This problem is called the *two-pin linear net routing* (TLNR) *problem*. Clearly, we do not need to introduce feedthroughs to connect any net in the TLNR problem. Moreover, the two possible ways of connecting a two-pin linear net correspond to the two net segments of the net. Therefore, the TLNR problem is a special case of the general net segment selection problem that we are interested. However, the TLNR problem was shown to be NP-hard if there are five or more cell rows in the design [15]. Thus, the net segment selection problem is NP-hard if there are five or more cell rows in the design. □

In the remainder of this paper, we present an efficient heuristic algorithm for computing a standard cell global routing solution.

3. The new standard cell global routing algorithm

Our global router works in two stages. In the first stage, we determine all the feedthroughs to be added and determine their locations within the rows. In the second stage, we choose which physical pins will be connected (and thus choose the net segments) to complete connections for each net.

3.1. Determine feedthroughs

In most previous algorithms, feedthroughs are added only when connections for some nets cannot be completed. In our algorithm, we use feedthroughs not only to complete the connections but also to trade off the width and height of the chip. Additional feedthroughs can often reduce track density and thus

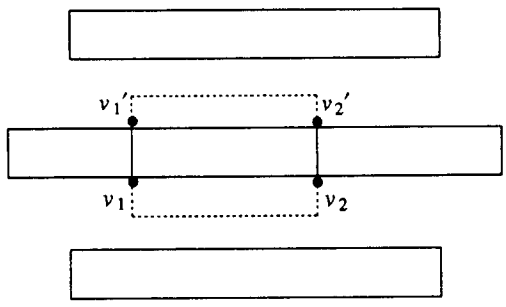


Fig. 4. Two possible ways to connect a two-pin linear net.

reduce chip height with no expense in width. An additional feedthrough on other than the longest row will not increase the width of the chip.

We generalize Kruskal's minimum spanning tree algorithm [17] to build a minimum spanning forest of the net connection graph. Feedthroughs are determined by the intersections of cell rows and the edges in the minimum weighted spanning forest.

We weight each edge according to the length of the edge and the cost to insert feedthroughs for the edge. For each edge $e = (p_i, p_j)$ in the net connection graph connecting two pins p_i and p_j in the same net, we define the weight of e

$$w(e) = |x_i - x_j| + K \cdot \sum_{e \cap R_i \neq \phi} \text{weight}(R_i)$$

where x_i and x_j are the horizontal coordinates for p_i and p_j, respectively. K is a constant factor. $e \cap R_i \neq \phi$ means that net edges e intersects row R_i, weight(R_i) is the weight of row R_i, which is based on the current length of row R_i. Assume we choose e to be included in the minimum weighted spanning forest. If p_i and p_j are in the same channel, we simply add e into the solution. If p_i and p_j are in different channels, we add feedthroughs $f_1, f_2, \ldots, f_l$ in rows $R_{i_1}, R_{i_2}, \ldots, R_{i_l}$ which intersect with e. Then we add the path from p_i to p_j through these feedthroughs into the solution. The cost of an edge thus defined is a function of both the wirelength and the cost of adding feedthroughs. By assigning different weights to different rows, we discourage adding feedthroughs in the longest rows since this will increase the width of the chip. On the other hand, when two pins are far apart we may connect them to nearby pins in the same net, even at the cost of extra feedthroughs. These extra feedthroughs may decrease the total channel density and thus decrease the height of the chip. We adjust the factor K to control the trade-off between the width and the height of a chip.

Note that the weight of each net edge is not static. After a feedthrough is added, some pin locations and the weights of some rows may change. If we construct the minimum spanning forest by building a minimum weighted spanning tree net by net, a different order for considering the nets will quite likely lead to a different result. However, there is no efficient algorithm available to determine an optimal net order. In order to avoid the net ordering problem, our algorithm builds a minimum spanning forest directly by considering all of the nets simultaneously based on a generalization of the Kruskal's minimum spanning tree algorithm [17]. It keeps adding the minimum weighted edge selected from the entire net connection graph into the spanning forest as long as no cycle is introduced. The edge insertion process ends when all the nets are connected. Our algorithm for the first stage is shown as follows, in which F represents the spanning fores to be constructed.

Algorithm 1. Determine_Feedthroughs (* Stage 1 of global routing *)

1. $V :=$ all the vertices in the net connection graph;
   $E :=$ all the edges in the net connection graph;
   $F := \varnothing$;

2. **while** $E \langle\rangle \varnothing$ do
   remove the minimum weighted edge e from E;
   if $F \cup \{e\}$ does not have a cycle
   then include e (or the path induced by e) in F;
   if e crosses some cell rows
   then introduce feedthroughs at the intersections and add them to V;
   introduce edges connected to the new feedthroughs and add them to E;
   end-while
3. output F

The advantage of this algorithm is clear. Since we consider the edges of all nets at the same time, the algorithm is independent of input net order. The spanning tree for each net gets an equal chance to grow. Information about all nets is available throughout the process.

It is necessary to show that this algorithm will converge since we keep adding new vertices (induced by feedthroughs) to the net connection graph. In [1], a limit is set on the total number of vertices allowed for each net. However, we can show that our algorithm ends after a linear number of edge insertions.

Theorem 2. *The Stage 1 algorithm will converge to a spanning forest after $n - k$ steps of edge insertions, where n is the total number of original pins in the design, and k is the number of nets.*

Proof. Let m be the number of steps of edge inclusion that we execute to obtain a spanning forest. Let V_i be the set of vertices in the net connection graph after the i-th step of edge inclusion. Let F_i be the set of edges in the partially constructed spanning forest after the i-th step of edge inclusion. Initially, $|V_0| = n$ and $|F_0| = 0$. When we obtain a spanning forest, we have k connected components and each of them is a spanning tree. Thus, $|V_m| - |F_m| = k$.

Let $e_i = (u_i, v_i)$ be the edge included in the i-th step. There are two cases: (i) If u_i and v_i are in the same channel, then $V_i = V_{i-1}$ and $F_i = F_{i-1} \cup \{(u_i, v_i)\}$; (ii) If u_i and v_i are in different channels, suppose that $f_1, f_2, \ldots, f_l$ are the vertices along the path introduced by adding feedthroughs, then $V_i = V_{i-1} \cup \{f_1, f_2, \ldots, f_l\}$ and $F_i = F_{i-1} \cup \{(u_i, f_1), (f_1, f_2), \ldots, (f_l, v_i)\}$. In both cases, we have

$$|V_i| - |F_i| = |V_{i-1}| - |F_{i-1}| - 1$$

By induction, it is easy to show that

$$|V_m| - |F_m| = |V_0| - |F_0| - m$$

It follows that $k = n - m$, i.e., $m = n - k$. □

In the implementation of Algorithm 1, we use the *union-find* operations [17]. Initially, we start with n sets. Each set contains a single vertex. When we include an edge (u, v), we union the two sets that vertices u and v belong to, together

with the set of new vertices introduced by adding feedthroughs. At any time of execution, each set contains a collection of vertices which have been connected. Cycle detection becomes very easy. To see whether $F \cup \{(u, v)\}$ contains a cycle, we need to only check whether u and v belong to the same vertex set. In fact, after each edge insertion, we remove all the edges connecting vertices of the same set. Moreover, we update all the edge costs after each edge insertion. Let p be the maximum number of pins per net (in CMOS technology, p is bounded by a small constant). Then, each edge insertion takes (kp^2) time. Since there are $n-k$ edge insertions and $O(kp) = O(n)$, the time complexity is $O(pn^2)$ for Stage 1 of our global routing algorithm.

Note that after Stage 1 of our algorithm, we actually obtain a global routing solution in which the feedthroughs are specified by the vertices in the spanning forest and the net segments are specified by the edges in the spanning forest. However, the selection of the net segments is made without consideration of global density distribution, and may lead to a poor routing solution in terms of minimizing the total channel density. Therefore, our algorithm will go through the second stage, as described in the next subsection, to re-compute the best net segment selection for total channel density minimization.

3.2. Determine net segments

After Stage 1, all of the required feedthroughs have been added and their positions have been determined; we shall not add new feedthroughs. Thus, we can remove those edges that cross the cell rows (but are not *built-in* edges that represent feedthroughs or connections within the cells) from the net connection graph. Figure 5 shows an example of a connected component induced by a net in the net connection graph at the beginning of Stage 2. The solution of Stage 2 is a spanning forest S of the net connection graph such that each edge in S lies entirely in one channel. Since the width of a chip is fixed after Stage 1, the objective in Stage 2 is to minimize the height of the chip by minimizing the total channel density.

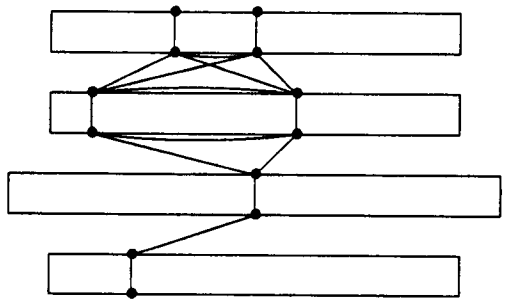


Fig. 5. The connected component induced by a net in the net connection graph at the beginning of Stage 2.

Most previous approaches to global routing build a spanning tree for each net one by one. And for each net, these algorithms keep adding the minimum weighted feasible edge until a spanning tree is obtained. A serious problem exists with these approaches. It is very difficult to decide whether and where a net segment should be added to a channel since these algorithms have no knowledge of the density distribution in the final solution, especially early in the execution of the algorithms when only few net segments are present. Also, these approaches face the problem of choosing a good order to process the nets.

To avoid these problems, we develop a new algorithm based on the iterative deletion approach. First, the basic approach is described; then we will show the performance improvements and refinements. The algorithm constructs a minimum weighted spanning forest to approximate the optimal spanning forest. We define the weight of an edge e to be $w(e) = d(e)/d$, where $d(e)$ is the maximum density over the edge in the channel to which e belongs, and d is the density of the channel. First, we put all the edges in the net connection graph into an edge set S. Then, we repeatedly remove the maximum weighted edge from S as long as we do not disconnect any net. We update the weights of edges in a channel whenever an edge is removed from that channel. The process terminates when S is a spanning forest. Clearly, this approach has two advantages:

(1) Since all the edges are considered from the start, the algorithm has global information and knows where the most congested areas are. The weight of each edge during the construction reflects the relative density over that edge in the resulting spanning forest, and the removal of the maximum weighted edges distributes the routing density evenly in the design;
(2) Since we process all nets in parallel, the result of our algorithm does not depend on the order in which nets are processed.

However, a straightforward implementation of the above algorithm may suffer two problems. First, there may be $\Theta(n^2)$ edges in the net connection graph at the beginning of Stage 2 (even after we removed edges, other than the built-in edges, which crossed cell rows), where n is the number of pins after Stage 1. Since there are only $O(n)$ edges in the final spanning forest, we may have to go through $\Theta(n^2)$ steps of edge deletion; which is quite time consuming. Moreover, since most of the initial edges are to be removed, the weight of an edge at the beginning of the deletion process may not closely approximate the channel density over that edge in the final spanning forest. Both problems are due to the fact that we may have to start with a quadratic number of edges at the beginning of the edge deletion process. A careful study showed that we can further simplify the net connection graph before we compute the optimal spanning forest. We define the *simplified net connection graph* SG to be an undirected graph, whose vertex set is the set of pins in the design, and for two pins p_i and p_j of the same net, (p_i, p_j) is an edge of SG if and only if the two pins satisfy one of the following conditions: (1) p_i and p_j are on the same cell or feedthrough and are connected internally (i.e., they are connected by a build-in edge); or (2) p_i and p_j are in the same channel and there are no other pins of the same net in the same channel between them. Figure 6 shows the example of a connected

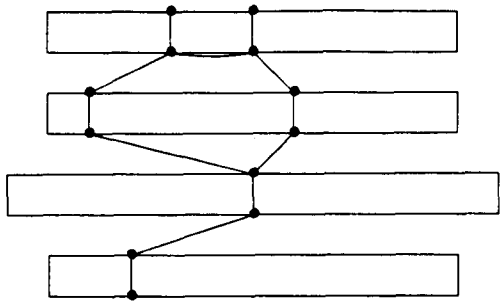


Fig. 6. The connected component induced by the same net in the simplified net connection graph is much sparse than the one in Fig. 5. The optimal spanning forest is not lost by this simplification.

component induced by a net in the simplified net connection graph. Note it is much more sparse than the one shown in Fig. 5. In fact, we can make the following claims:

Theorem 3. *Let n and m be the number of vertices and edges, respectively, in the simplified net connection graph. Then*

(1) $m \leqslant 1.5n$.

(2) *The simplified net connection graph can be constructed in* $O(n \log n)$ *time, where n is the total number of pins in the design.*

(3) *The simplified net connection graph contains an optimal spanning forest.*

Proof. (1) For each vertex v, the degree $d(v)$ of v in the simplified net connection graph is at most 3, since it can be connected to only its left closest pin in the same net, its right closest pin in the same net, and its equivalent pin in the same cell (see Fig. 6). We have

$$2m = \sum_{i=1}^{n} d(v_i) \leqslant 3n$$

Thus, $m \leqslant 1.5n$.

(2) The simplified net connection graph can be constructed by sorting pins in each channel, then do a linear scan to compute the left closest pin and the right closest pin in the same net for each pin in the channel. The total scanning time is bounded by $O(\sum_{i=1}^{k} n_i)$, where k is the number of nets and n_i is the number of pins in channel i. Note that $\sum_{i=1}^{k} n_i = n$, thus, the complexity is dominated by the total sorting time, which is

$$\sum_{i=1}^{k} n_i \log n_i \leqslant \left(\sum_{i=1}^{k} n_i \right) \log \sum_{i=1}^{k} n_i = n \log n$$

(3) Let F be an optimal spanning forest. Suppose $e = (u, v)$ is an edge in F but e is not in the simplified net connection graph, without loss of generosity, assume v is right to u. Let $w_1, w_2, \ldots, w_i$ be all the pins of the same net in the same channel from left to right between u and v. Clearly, (u, w_1), $(w_1,$

$w_2),\ldots,(w_l, v)$ are all in the simplified net connection graph. It is not difficult to show that we can remove (u, v) and add a set of edges (probably all) of (u, w_1), $(w_1, w_2),\ldots,(w_l, v)$ to make another spanning forest F'. It is clear that $d(F') \leqslant d(F)$. Thus, F' is also an optimal spanning forest. By repeating this process, we can remove all the edges in F but not in the simplified net connection graph to obtain another optimal spanning forest which contains edges only from the simplified net connection graph. □

The benefits from the theorem are clear. Since we only have to go through approximately $0.5n$ number of edge deletions, our algorithm runs much faster. Also, since only a relatively small number of edges in SG are to be removed, the weight of each edge measures more accurately the density over the edge in the resulted spanning forest. Moreover, we can compute the simplified net connection graph efficiently without losing the optimal spanning forest. We summarize our algorithm for Stage 2 as follows:

Algorithm 2. Determine the Net Segments (* Stage 2 of global routing *)
1. build the the simplified net connection graph;
2. $S :=$ all the edges in the simplified net connection graph;
3. **repeat**
   Remove the maximum weighted edge in S that is in a cycle;
   Update edge weights for the affected edges;
   until S is a spanning forest;
4. outout S.

In our implementation, we use the graph biconnectivity algorithm in [18] to identify all the edges which are in some cycles. An edge is in some cycle if and only if it belongs to a biconnected component with no less than 3 vertices. We can generate all the biconnected components of a graph in linear time using the depth first search algorithm. Moreover, after removing an edge, we need to update the weights of all the edges in the same channel. We may have $\Theta(n)$ edges in the channel in the worst case, and a straightforward computation of each edge weight takes $O(L)$ time, where L is the number of physical pins in the channel. Thus, updating all the edge weights takes $O(nL)$ time, and the time complexity of the second stage of our algorithm is $O(n^2L)$.

For large designs, we can use a data structure called *segment tree* [19] to reduce the time for updating an edge weight from $O(L)$ to $O(\log L)$. For each channel C, we construct a segment tree T_c so that the maximum density over any given interval in the channel can be computed in $O(\log L)$ time. Without loss of generality, we assume that $L = 2^l$ for some l. Let $x_1, x_2,\ldots, x_L$ be the x-coordinates of the pins in the channel C. The segment tree T_C is a balanced binary tree with L leaves in which each leaf corresponds to the x-coordinate of a pin (i.e., an interval of length zero) and each internal node (root of a subtree) corresponds to an interval (from the leftmost leaf to the rightmost leaf in the subtree). Clearly, the root of the i-th subtree of height j corresponds to the

interval $[x_{(i-1)\cdot 2^l+1}, x_{i\cdot 2^l}]$. Such an interval is called a *power interval*. Each node X in the tree has three fields: $X\_interval$ stores the corresponding power interval, $X\_density$ stores the maximum density over the power interval *X-interval*, and X_delete indicates how many times that $X\_interval$ is deleted from the subtree rooted at X. At the beginning of the iterative deletion algorithm, we compute $X\_density$ for each node in the tree T_c and set X_delete to be zero. For example, Fig. 7(a) shows a set of intervals and the corresponding segment tree. Suppose that we want to remove an edge in channel C whose interval is I. It is not difficult to show that I can be decomposed into O(log L) maximal power intervals $I_1, I_2, \ldots, I_t$, which correspond to a set of nodes $X_1, X_2, \ldots, X_t$ in the segment tree. For example, in Fig. 7(a), interval $J_5 = [5,7]$ can be decomposed into intervals [5,6] and [7,7], and interval $J_8 = [3,8]$ can be decomposed into intervals [3,4] and [5,8]. For each I_i, we make the following updates:

$$X_i\_delete = X_i\_delete + 1 \quad \text{and} \quad X_i\_density = X_i\_density - 1, \quad 1 \leqslant i \leqslant t$$

Moreover, we update the nodes on the paths from the root to X_i's. Let Y_i be the sibling of X_i and Z_i be the parent node of X_i. Then,

$$Z_i\_density = \max(X_i\_density, Y_i\_density) - Z_i\_delete$$

and the update of the *density* field propagates upward in the tree. We update the density field of all the relevant nodes on the l-th level first before we move to the $(l-1)$-th level. For example, Fig. 7(b) shows the corresponding segment tree after we remove intervals J_5 and J_8 from Fig. 7(a). In order to compute the maximum density over a given interval I, we, again, decompose I into O(log L) maximal power intervals $I_1, I_2, \ldots, I_t$. Let $X_1, X_2, \ldots, X_t$ be the corresponding nodes in the tree. Let P_i be the path in the tree T_c from the root to the parent node of X_i. Then, the maximum density over the interval I_i is

$$d[I_i] = X_i\_density - \sum_{Y \in P_i} Y\_delete,$$

and the maximum density over I is $\max_{i=1}^{t} d[I_i]$. For example, the maximum density over the interval J_2 in Fig. 7(b) is

$$\max(d[4], d[5,6]) = \max(3-1, 3-1) = 2$$

Therefore, we can update each edge weight in O(log L) time after we remove an edge from the channel. With this refinement, the second stage of our global routing algorithm can be implemented in O(n^2 log L) time.

4. Experimental results

We implemented our algorithm in the Cedar language running on Xerox Dorado workstations (2-MIPS machines) and incorporated it into the DATools system developed at Xerox PARC. Table 1 summarizes the examples used to compare the new algorithm with the previous global router in the DATools

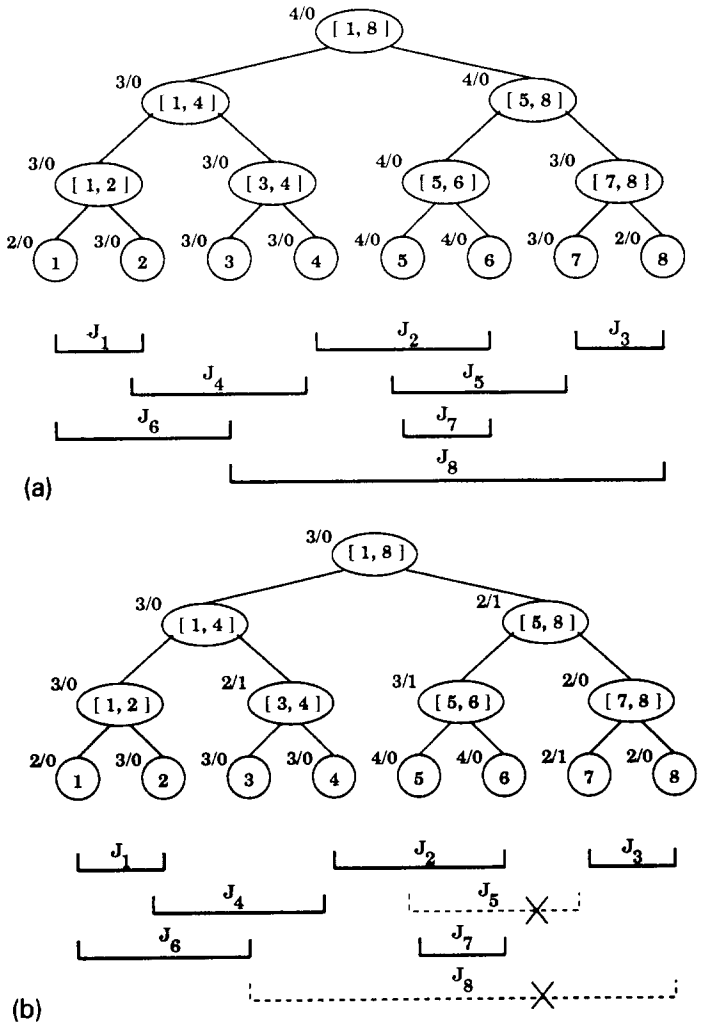


Fig. 7. (a) A set of intervals and its corresponding segment tree. The labels at each node X specify $X\_density / X\_delete$, respectively. (b) The new segment tree after intervals J_5 and J_8 are deleted.

Table 1
A summary of the example circuits used to compare the new algorithm with the previous algorithm and with the global router in Timberwolf 4.2

Example	#cells	#IOS	#nets	#pins
16-bit adder	144	50	177	546
16-bit counter	173	56	206	609
32-bit adder	288	98	355	1090
32-bit counter	342	104	396	1203
64-bit adder	576	194	707	2178
64-bit counter	681	200	783	2393
Primary 1	752	81	904	2737
Primary 2	2907	107	3029	8758

Table 2
Comparisons with the previous global routing package in the Xerox PARC DATools system

Example	#of rows	previous algorithm		new algorithm		improvement
		width	height	width	height	
16-bit adder	4	1104	812	1104	764	6.3%
16-bit counter	5	1320	1120	1320	1008	11.1%
32-bit adder	6	1528	1415	1528	1324	6.8%
32-bit counter	7	1904	1736	1904	1624	8.5%
64-bit adder	8	2448	1956	2448	1860	5.1%
64-bit counter	9	3096	2744	3096	2456	11.7%

system and with the global router in Timberwolf 4.2. The counters and adders are the circuits synthesized by the DATools system when no performance requirements are imposed. Both types of circuits are simple, ripple-carry designs. Primary 1 and Primary 2 are the benchmarks from the Physical Design Workshop [2].

Table 2 summarizes the experiments comparing the old and new algorithms. Compared to the previous standard cell global routing package used in the DATools system, the new algorithm achieves a 6 to 11% area reduction. In general, more feedthroughs are added by the new algorithm, but the chip widths are not increased. The extra feedthroughs are being used to reduce chip height by reducing total track density.

We also compared our global routing results with the results produced by the Timberwolf 4.2 global routing on the two Physical Design Workshop benchmarks. In both examples, the global routing is performed on the *same placement* (the one produced by Timberwolf). Figure 8 illustrates the comparison process. Table 3 shows the comparisons on these examples. We obtained 5 to 17% reduction in total track density and over 20% reduction in the number of inserted feedthroughs compared to Timberwolf 4.2. We could not compare our algorithms with other algorithms since we were unable to obtain the common placement solutions. (We observed that the placement solution affects both the number of feedthroughs and the total channel density significantly. Thus, it

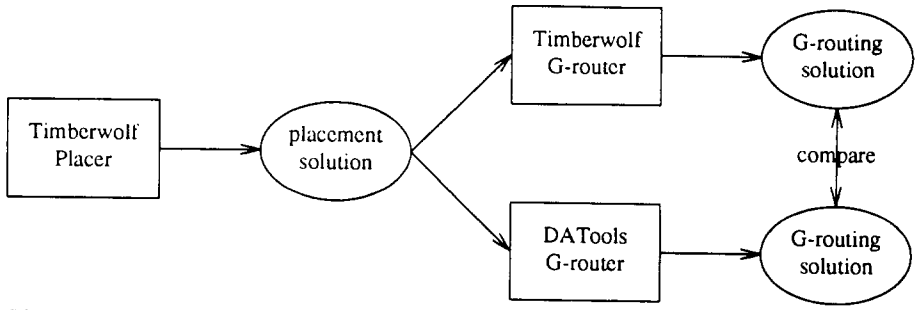


Fig. 8. Comparison of global routing solutions is based on the same placement solution.

Table 3
Comparisons with the global routing algorithm in Timberwolf 4.2 [14] on Primary 1 and Primary 2 benchmarks from the Physical Design Workshop

Example	#of rows	Timberwolf		new algorithm		improvement	
		#FTs	#tracks	#FTs	#tracks	#FTs	#tracks
P1	17	1380	223	1120	190	23.2%	17.4%
P2	29	4621	474	3761	449	22.9%	5.6%

would be very inaccurate to compare global routing results based on different placement solutions.)

Our global router is a straightforward implementation of the algorithm described here. As a result, there are a number of opportunities for further improvement of the results. Some standard cell global routing packages improve the global routing (and further reduce area) by exchanging adjacent cells in the same row or modifying the cell orientation [1,4,14]. In addition, the cells in the Physical Design Workshop Benchmarks (Primary 1 and Primary 2) have a large number of built-in feedthroughs that are exploited by Timberwolf, but not by the global router described here. (The standard cells used at Xerox PARC do not have built-in feedthroughs so this feature was not included.) We did not implement these refinements in the current version of our global routing package.

5. Remarks and conclusions

In this paper, we present a new algorithm for standard cell global routing. By processing all the nets in parallel, we avoid the problems associated with net ordering and the problems created by lack of congestion information early in the global routing process. By simplifying the net connection graph and applying an iterative deletion algorithm for building spanning trees, we can more accurately predict congested areas. This global routing algorithm produces high quality solutions in polynomial time.

In both Stage 1 and Stage 2, the weights for edges in the net connection graph are dynamic. A significant amount of time is spent on updating edge weights in our current implementation. We are studying how to reduce the computation time for updating edge weights. One possibility is to use more sophistical data structures to identify those edges whose weights need to be updated more efficiently (without scanning all the edges). This will certainly speed up the re-computation for edge weights. Another possibility is not to update the edge weights after every iteration but to update them after several iterations. Our experience is that small errors in the edge weights will not affect the quality of final solution significantly.

Acknowledgment

This research is partially supported by the National Science Foundation under grant MIP-9110511.

References

[1] Roberts, K.A., Automatic layout in the Highland system, *Proc. of Int. Conf. on Computer-Aided Design* (1984) pp. 224–226.
[2] Preas, B., Benchmarks for cell-based layout systems, *Proc. of 24th Design Automation Conf.* (1987) pp. 319–320.
[3] Supowit, K.J., Reducing channel density in standard cell layout, *Proc. of 20th Design Automation Conf.* (1983).
[4] Preas, B., The standard cell package, Xerox PARC internal document, 1986.
[5] Barth, R., L. Monier, and B. Serlet, PatchWork: layout from schematic notations, *Proc. of 25th Design Automation Conf.* (1988) pp. 250–255.
[6] Mowchenko, J.T. and C.S.R. Ma, A new global routing algorithm for standard cell ICs, *Proc. of IEEE International Symposium on Circuits and Systems* (May 1987) pp. 27–30.
[7] Lee, K.W. and C. Sechen, A new global router for row-based layout, *Proc. of Int.'l Conf. on Computer-Aided Design* (1988) pp. 180–183.
[8] Rose, J., LocusRoute: a parallel global router for standard cells, *Proc. of 25th Design Automation Conf.* (1988) pp. 189–195.
[9] Brouwer, R. and P. Banerjee, Phigure: a parallel hierarchical global router, *Proc. of 27th Design Automation Conf.* (1990) pp. 650–653.
[10] Miexner, G. and U. Lauther, A new global router based on a flow model and linear assignment, *Proc. of Int. Conf. on Computer-Aided Design* (1990) pp. 44–47.
[11] Aoshima, K. and E.S. Kuh, Multi-channel optimization in gate-array LSI layout, *Proc. of IEEE International Symposium of Circuits and Systems* (1983) pp. 1005–1008.
[12] Lin, L., S. Sahni and E. Shragowitz, An enhanced heuristic for multi-channel optimization in gate-array layout, *Proc. of Int. Conf. on Computer-Aided Design* (1986) pp. 242–244.
[13] Cong, J. and B. Preas, A New Algorithm for Standard Cell Global Routing, *Proc. of Int. Conf. on Computer-Aided Design* (1988) pp. 176–179.
[14] Sechen, C. and A. Sangiovanni-Vincentelli, Timberwolf3.2: a new standard cell placement and global routing package, *Proc. of 23rd Design Automation Conf.* (1986) pp. 432–439.
[15] Blair, J., S. Kapoor, E. Lloyd, and K. Supowit, "Minimizing channel density in standard cell layout, *Algorithmica* **2** (1987) 267–282.
[16] Kapoor, S., *Topics in the design and analysis of combinatorial algorithms*, Ph.D. Thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, 1986.
[17] Aho, A., J.E. Hopcraft and J.D. Ullman, *The Design and Analysis of Computer Algorithms* (Addison-Wesley, Reading, MA, 1974).
[18] Reingold, E.M., J. Nievergelt and N. Deo, *Combinatorial Algorithms: Theory and Practice* (Prentice-Hall, Englewood Cliffs, NJ, 1977).
[19] Preparata, F.P. and M.I. Shamos, *Computational Geometry* (Springer, New York, 1985).