VLSI DESIGN
1996, Vol. 5, No. 1, pp. 37–48
Reprints available directly from the publisher
Photocopying permitted by license only

A Fast Clustering-Based Min-Cut Placement Algorithm with Simulated-Annealing Performance

YOUSSEF SAAB*

Department of Computer Engineering and Computer Science, University of Missouri-Columbia, Columbia, MO 65211

(Received 6 September 1994; In final form 30 January 1995)

Placement is an important constrained optimization problem in the design of very large scale (VLSI) integrated circuits [1–4]. Simulated annealing [5] and min-cut placement [6] are two of the most successful approaches to the placement problem. Min-cut methods yield less congested and more routable placements at the expense of more wire-length, while simulated annealing methods tend to optimize more the total wire-length with little emphasis on the minimization of congestion. It is also well known that min-cut algorithms are substantially faster than simulated-annealing-based methods. In this paper, a fast min-cut algorithm (ROW-PLACE) for row-based placement is presented and is empirically shown to achieve simulated-annealing-quality wire-length on a number of benchmark circuits. In comparison with Timberwolf 6 [7], ROW-PLACE is at least 12 times faster in its normal mode and is at least 25 times faster in its faster mode. The good results of ROW-PLACE are achieved using a very effective clustering-based partitioning algorithm in combination with constructive methods that reduce the wire-length of nets involved in terminal propagation.

Keywords: Row-based placement, min-cut placement, standard-cell placement, partitioning, simulated annealing, layout

1. INTRODUCTION

The design of VLSI circuits is a complex process that transforms a design specification into a physical circuit through several inter-dependent steps. The layout problem is an important stage of the overall design process and it involves the assignments of geometric locations to the elements of the circuits and the electric wire connections among them. Due to an enormous combinatorial complexity, the layout problem has been traditionally performed in two stages. In the first stage, called placement, all the circuit elements are assigned fixed geometric locations on the layout surface. The second stage, called routing, consists of the physical realization of the connections among the elements of the circuit subject to a technology-dependent set of constraints. Placement and routing are inter-dependent and better layouts may be

*Phone: (314)-882-4559. Fax: (314)-882-8318. E-mail: saab@ysaab.cs.missouri.edu.

achieved by performing placement and routing simultaneously. There have been previous efforts at combining placement and routing [8–11], but the ever increasing size of electronic circuits is rendering the placement followed by routing the more practical approach to the layout problem. Routing is highly dependent on the placement stage. A good placement simplifies the subsequent routing step, while a bad placement may render routing an impossible task. Therefore, routability is a major goal of placement among many other goals such as minimizing timing delays on critical nets, maximizing circuit performance, and minimizing layout area.

Successful placement algorithms have been obtained using general optimization paradigms such as simulated annealing [5,7] and genetic algorithms [12–14]. Other existing placement techniques are:

Incremental construction: The strategy used here is to place the nodes of the circuit successively until all of them have been placed. Seed nodes are placed first. Subsequently, based on a selection rule, an unplaced node is chosen and is placed in the next best vacant position. This process is repeated until all the circuit elements have been placed, approaches of this kind are reported in [15].

Node exchange: This is an iterative improvement approach in which an initial placement is improved by the exchange of the positions of some of the nodes. A widely used strategy is the exchange of the positions of two nodes until no further improvements can be made [16,17].

Combinatorial methods: Branch-and-bound is a strategy that systematically explores the solution space of a combinatorial problem in search of the optimal solution [3]. The strategy is used to prune or reduce the number of placements explored in what would otherwise be a complete exhaustive search of all possible placements. Branch-and-bound algorithms tend to be computationally expensive and are not used except for problems of small size [1]. Branch-and-bound techniques have been reported in [15].

Analytical methods: It is possible to formulate the placement problem as a non-linear mathematical problem, where non-linear programming techniques are applicable [18,19].

Min-cut placement: Methods in this class place the nodes of the circuits by a recursive use of a partitioning method. Basically, the circuit is partitioned into two parts. The available layout area is then cut by a straight line into two parts on each side of the cutting line. Each of the two parts of the circuit is then assigned to the two parts of the layout area. This gives rise to two smaller placement problems which are then recursively solved by the same method until each sub-circuit consists of one cell [6,11,20].

Force-directed placement: Methods in this class view the connections among circuit nodes as binding forces that are trying to keep the connected nodes in close proximity. Therefore, a node is in a good location if the total force exercised on it, is zero. A common denominator of these methods is the determination of the best location for a node. Usually, nodes are successively moved in some order to their ideal locations in an effort to optimize the placement. Such methods are reported in [21].

Spectral methods: These methods use a mathematical formulation of the placement problem, where placement properties are usually related to the eigenvalues of an associated matrix. These relationships are exploited to design of approximation placement algorithms. Several spectral approaches have been reported in the literature [22,23].

Resistive network optimization: This approach has been proposed by Cheng and Kuh [24]. Basically, the placement problem is formulated as the problem of minimizing the power dissipation in a resistive network. A solution of the resistive network is then translated into a placement of the circuit.

Among all placement methods, simulated annealing is currently the most popular and is the best algorithm available in terms of the placement quality, but it is too time consuming. One of the best available simulated annealing placement package is Tim-

berwolf 6 [7]. Min-cut algorithms rank second to simulated annealing in terms of placement quality but are substantially much faster [1,4]. The contribution of this paper is the design of a min-cut algorithm (ROW-PLACE) with results that are competitive with Timberwolf 6 in terms of quality. In terms of speed, ROW-PLACE in its normal mode is at least 12 times faster than Timberwolf 6, and, in its fast mode, ROW-PLACE is at least 25 times faster than Timberwolf 6. ROW-PLACE is distinguished from previous min-cut placement methods by an effective clustering-based partitioning algorithm in combination with constructive methods that reduce the wirelength of nets involved in terminal propagation.

2. ROW-BASED PLACEMENT

For placement purposes, an electrical circuits consists of a hypergraph along with geometric descriptions of its components. A hypergraph $G(V, E)$ consists of a set of nodes V and a set of nets E. Each net $e \in E$ is a subset of 2 or more nodes in V. In the hypergraph model of an electrical circuit, each node correspond to a component of the circuit, and each net represents a common electrical signal among its constituent nodes. A *pin* is a point of contact of a net with one of its constituent nodes. A net may touch a node in more than one pin. The locations of pins of a node are specified by relative coordinates with respect to the center of that node. The nodes are usually rectangular in shape and are placed so that their sides are parallel to the reference coordinate axes in the plane. Therefore, the location of a node is completely specified by the coordinate of its center if only one orientation of the node is allowed.

Row-based placement is an approach applicable to design styles such as standard cells, gate arrays, and field programmable gate arrays. In this approach, the nodes of the circuit have a common height but differ in length, and they can be placed in horizontal rows, where each row has the same common height of the nodes. The space between rows is reserved for routing. The number of rows is a user-chosen parameter and is usually chosen so that the layout space used is approximately a square. The length of a row is the sum of lengths of nodes assigned to it. Therefore, to avoid wasted space at the end of short rows, the placement algorithm must balance the lengths of rows.

3. OUTLINE OF ROW-PLACE

Roughly, the min-cut approach used is the same as in [20]. We alternate the partitioning of the circuit nodes by vertical and horizontal lines until the nodes are localized in small areas where they can be assigned to specific locations in specific rows. To ease the description of the process, let us define a *rectangulation* of the layout surface to consist of:

1) A partition of the rows into horizontal slabs. Each slab consists of one or more consecutive rows and each row belongs to a unique slab.
2) Each slab is in turn partitioned into an ordered sequence of rectangles by means of vertical lines.

Consider a slab S of a rectangulation that spans rows l to h and consists of k rectangles $r_1,\ldots, r_k$ ordered from left to right. Let $m = [(l + h)/2]$. The slab S can now be partitioned into two slabs U and D by a horizontal line that cuts each rectangle r_i in S into two rectangles u_i and d_i. Slab U spans rows $m + 1$ to h and consists of k rectangles $u_1,\ldots, u_k$, and slab D spans rows l to m and consists of k rectangles $d_1,\ldots, d_k$. A rectangulation can be refined by cutting each of its slabs with more than one row by a horizontal line. Call this operation a *y-refinement*. A slab S that consists of k rectangles $r_1,\ldots, r_k$ can be refined by cutting each one of its rectangles in half by a vertical line. Let x_i and y_i be the left and right halves of rectangle r_i. The refined slab consists of $2k$ rectangles $x_1, y_1,\ldots, x_k, y_k$. An *x-refinement* is the application of the above operation to each slab of the rectangulation.

A placement of a circuit into a rectangulation consists of assigning each of its nodes to a unique rectangle. The *length* of a rectangle residing in a slab that spans rows l to h is equal to the sum of all the lengths

of the nodes assigned to it divided by $h - l + 1$. The x-coordinates of rectangles are computed according to the order of rectangles in their respective slabs from left to right. Consider a slab of k rectangles $r_1,\ldots, r_k$ that are ordered from left to right. Let x_i be the x-length of rectangle r_i. The x-coordinate of rectangle r_i is $x_i/2 + \sum_{j=1}^{i-1} x_j$. Thus, r_1 has $x_1/2$ as its x-coordinate, r_2 has $x_2/2 + x_1$ as its x-coordinate, and so on. Each row in the placement is at some y level. Let h be the common height of all the nodes of the circuit. Then row i has the y-coordinate $h/2 + 2(i - 1)h$. Thus, it is assumed as in Timberwolf [7] that adjacent rows are separated by distance h. This assumption is only used to estimate the wire-length and the separation between adjacent rows can only be determined after routing. The y-coordinate of a slab is the average y-coordinate of its rows. Each rectangle of the slab has the same y-coordinate as the slab it belongs to. The x and y coordinates of a node in a placement are those of its enclosing rectangle. The length of a net in a placement is the half-perimeter of the rectangle that encloses all its pins. The wire-length of the placement is the sum of lengths of all nets.

A placement into a rectangulation can be refined by applying either a y-refinement or an x-refinement. When a rectangle is cut into two rectangles, the nodes assigned to it are partitioned between the two resulting rectangles so that the two new rectangles are about equal in length.

The initial rectangulation is one that consists of one slab that spans all the rows and that consists of one rectangle. By repeated applications of x-refinements and y-refinements, we reach a rectangulation where each rectangle contains one node of the circuit and each slab spans one row. At this stage, the coordinates of each node specify the location of that node in the final placement. Many different sequences of x-refinements and y-refinements have been tried. The best approach was the one that alternates between the two refinements as in [20].

Figure 1 shows a placement of a small circuit obtained using ROW-PLACE. The length of nodes 1 through 8 are 22, 18, 30, 10, 15, 25, 35, and 5 respectively. The common height of the nodes is 5. Nets are

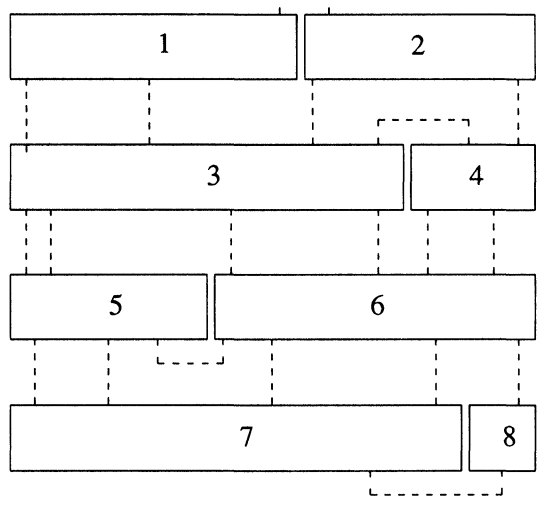

FIGURE 1 A small circuit placed by ROW-PLACE.

represented by dashed lines connecting the nodes. The sequence of rectangulations that led to the placement in Figure 1 is shown below. Each rectangulation is shown as a sequence of slabs. The rows spanned by each slab are indicated. Rectangles in each slab are listed from left to right as sets of nodes.

Initial rectangulation:

slab(row 1, row 4): {1, 2, 3, 4, 5, 6, 7, 8}

After y-refinement:

slab1(row 3, row 4): {1, 2, 3, 4}

slab2(row 1, row 2): {5, 6, 7, 8}

After x-refinement:

slab1(row 3, row 4): {1, 3}, {2, 4}

slab2(row 1, row 2): {5, 7}, {6, 8}

After y-refinement, the final rectangulation is:

slab1.1(row 4, row 4): {1}, {2}

slab1.2(row 3, row 3): {3}, {4}

slab2.1(row 2, row 2): {5}, {6}

slab2.2(row 1, row 1): {7}, {8}

4. TERMINAL PROPAGATION

Some of the earlier min-cut algorithms partitioned blocks of nodes without considering connections to other blocks. This scheme was inadequate because the nets that enter a block from the outside have an effect on where the elements of this block ought to be placed. Terminal propagation is a technique that allows each block of the circuit to remember and to include the effect of its connections with other blocks. This technique was first introduced by Dunlop and Kernighan [20], but it required the computation of a rectilinear Steiner tree. In ROW-PLACE, a simple terminal propagation approach is used.

Given a hypergraph $G(V, E)$ and a subset $X \subseteq V$, let $G[X]$ denote the sub-hypergraph of G induced by X. $G[X]$ itself is a hypergraph with node set X and net set $E_x = \{N \cap X: N \in E \text{ and } |N \cap X| > 1\}$. When a rectangle R is being cut into two rectangles U and D, the set X of nodes in R must be partitioned between U and D. This is accomplished by a partitioning algorithm applied to the sub-hypergraph $G[X]$ of the circuit hypergraph G. Terminal propagation is included by adding two artificial nodes u and d of zero length to the node set of $G[X]$. Node u (d) is to remain locked in U (D) during the partitioning of $G[X]$. Nets that have nodes outside rectangle R are biased toward U or D. Let N be a net of the circuit that has nodes in common with rectangle R. Let m and M be the minimum and maximum of the coordinates of nodes in N in the perpendicular direction to the cut line. Let m^* and M^* be two parameters chosen as we shall explain later. If both $m < m^*$ and $M^* < M$ then net $N \cap X$ is not included in the net set of $G[X]$. Otherwise, $(N \cap X) \cup B$ is included in the net set of $G[X]$, where B is equal to $\emptyset$, $\{d\}$ or $\{u\}$ according to whether $m^* \leq m \leq M \leq M^*$, $m < m^* \leq M \leq M^*$, or $m^* \leq m \leq M^* < M$. If the cut line is horizontal, then $m^* = M^* = y$, where y is the y-coordinate of rectangle R. If the cut line is vertical, then $m^* = x - l/8$ and $M^* = x + l/8$, where x and l are respectively the x-coordinate and the length of rectangle R. Basically, m^* and M^* are the levels of two lines on either side of the cut line and parallel to it. Let S denote the area between these two lines. A net in $G[X]$ is either deleted, biased to the appropriate side, or not biased according to whether it has nodes on both sides of S, only one side of S, or exclusively inside S. In the case of a vertical cut, m^* (M^*) is the mid-point of the segment between the center of rectangle R and the target center of rectangle D (U). The above values chosen for m^* and M^*, in addition to being the most intuitive and natural for this type of placement, have empirically been verified to be the best after experimentations with several different other choices. Because nodes have equal heights but different lengths, it was necessary to include a buffer zone in the case of a vertical cut but not in the case of a horizontal cut. In the case of a vertical cut, a node close to the cut line is not allowed to bias the partitioning process one way or another because it may end up on the other side of the cut line due to the different lengths of nodes. This does not happen for horizontal cuts, since a node, once assigned to one side of a horizontal cut line, stays on this same side for the remainder of the placement process.

5. THE PARTITIONING ALGORITHM

The input to the partitioning algorithm considered here is a hypergraph $G(V,E)$ with integer node sizes. Define the size $S(A)$ of a subset $A \subseteq V$ as the sum of the sizes of its constituent nodes. A *partition* of $G(V,E)$ is an **ordered** pair (U,D) such that $U \cup D = V$ and $U \cap D = \emptyset$. A partition (U,D) is *feasible* if $|S(U) - T| \leq M$, where M is the maximum node size in V and T is a desired target size for U. In ROW-PLACE, T is chosen so that a rectangle is cut into two rectangles of about equal length. The input hypergraph may contain one or both of the two bias nodes u and d of zero length that are designated to stay locked in U and D respectively during the partitioning process.

Compaction is an operation in which subsets of nodes are coalesced (compacted or clustered) into a single node each. When a subset of nodes $X \subseteq V$ of a hypergraph $G(V,E)$ is compacted into single node x, the nets incident with node x in the new hypergraph

are of the form $(e \cap (V - X)) \cup \{x\}$, where $e \in E$ is a net originally incident with some node in X, and with the provision that nets that are reduced to one node are discarded. All other nets and nodes in $V - X$ remain the same.

The algorithm, BISECT, briefly presented here is described in [25] and in more detail in [26]. BISECT uses information collected during iterative improvement to incorporate compactions of nodes in a dynamic way. In [25,26], it is empirically shown that BISECT results can be up to 73 times better than the Fiduccia-Mattheyses algorithm (FM). The good empirical performance of ROW-PLACE are mainly due to the highly effective partitioning algorithm BISECT.

Let *BISECT_AND_COMPACT*(G, P_1, P_2, G', P'_1, P'_2) be a function that takes as input a hypergraph G (V, E) along with an initial feasible partition (P_1, P_2) of G, and outputs a compacted hypergraph G' (V', E') along with a feasible partition (P'_1, P'_2) of G'. The following is a pseudo-code of

BISECT:

Generate an initial feasible partition (U, D) *of* G;
save (U, D) *as the best current feasible partition*;
WHILE (*improvements are made*) *DO* {
  let H *be a copy of the hypergraph* G;
  let (P_1, P_2) *be the best current feasible partition*;
  WHILE (*improvement or compactions are made*) *DO* {
    BISECT_AND_COMPACT(H, P_1, P_2, H', P'_1, P'_2);
    set $H = H'$ *and* (P_1, P_2) = (P'_1, P'_2);
    }
  compute the new best feasible partition of G *from* (P_1, P_2);
  }

Let (P_1, P_2) be a partition (not necessarily feasible) of a hypergraph G (V, E). Define B (i) to be the subset of the partition that contains node i for each $i \in V$, and let $\bar{B}$ (i) = $V - B$ (i) be the other subset. The cost of a partition (U, D) is the number of nets cut and is denoted by *cost* (U, D). The gain of a node i is defined as:

$$g(i) = cost(B(i), \bar{B}(i)) - cost(B(i) - \{i\}, \bar{B}(i) \cup \{i\}),$$

which is the decrease in cost due to moving node i from $B(i)$ to $\bar{B}$ (i).

The function *BISECT_AND_COMPACT* performs the following steps:

1) Free all nodes and set $c = 0$.
2) *Forward move*: Of the subsets P_1 and P_2, select the one that has excess size, call it F, and call the other subset T. Move a sequence of nodes $f_1,\ldots,f_k$ from F to T using a highest-gain-first scheme until either F is out of free nodes or a **stopping criterion** is satisfied. Lock $f_1,\ldots,$ and f_k in T, set $c = c + 1$, and let $L_c = \{f_1,\ldots,f_k\}$.
3) *Restore balance*: If the size of P_1 is equal to its target size then do nothing. Otherwise, of P_1 and P_2, call F the subset that has excess size, and call the other subset T. Move a sequence of nodes $r_1,\ldots,r_j$ from F to T using a highest-gain-first scheme until either F is out of free nodes or the size balance cannot be improved. Lock $r_1,\ldots,$ and r_j in T, set $c = c + 1$, and let $L_c = \{r_1,\ldots,r_j\}$.
4) *Save*: If the current partition is an improved feasible partition then save it.
5) *Repeat*: If there are still free nodes then go to Step 2.
6) *Compaction*: Compact G by coalescing together **some connected subsets** of $G[L_1]$, $G[L_2]$,..., $G[L_c]$.

BISECT is the main feature of our min-cut approach. However, in order not to make this paper unnecessarily long, the reader is referred to [25,26] in which the specific details of the implementation of BISECT are discussed at length. This leaves enough space to discuss additional relevant details of our implementation of ROW-PLACE without having to duplicate material available elsewhere. In the remainder of this section, the methods used to generate the initial feasible partition are presented.

Random initial feasible partition: Here each node is placed with equal probability in U or in D until one

of U or D exceeds its target size. At this point, the remaining nodes are put in the other subset that is not yet filled to capacity. As an exception, the first two nodes are placed in U and D respectively to guarantee that both subsets of the partition are not empty. This initial partitioning scheme is used when the input hypergraph does not contain any of the bias nodes.

One-sided construction: This method is used to generate the initial feasible partition when the input hypergraph has only one bias node. In this case the nodes are successively put in either U or D depending on whether the bias node is u or d. Since both cases are similar, assume without loss of generality that the bias node is u. Let T be the desired target size of U. Initially the bias node u is put in U. Then other nodes are consecutively added to U until the target size T is reached or is exceeded. At this point, the remaining nodes are put in D. The nodes are put in U using a highest-gain-first scheme, where the gain of a node is the number of nets connecting it to U. To guarantee that both U and D are not empty, the last node is assigned to D even if U has not yet been filled to capacity.

Two-sided construction: This method is used to generate the initial feasible partition when the input hypergraph contains the two bias nodes u and d. Initially the bias node u is put in U, and the bias node d is put in D. Then nodes are alternatively put in U and D until one of the subsets U or D reaches or exceeds its target size. At this point, the remaining nodes are put in the other subset that is not yet filled to capacity. In this initial partitioning scheme, the gain of a node is the number of nets connecting it to U minus the number of nets connecting it to D. The next node to be added to U has the highest current gain among all remaining unassigned nodes. The next node to be added to D has the least current gain among all remaining unassigned nodes.

One-sided and two-sided constructions of the initial partition, are used to reduce the wire-length of nets involved in terminal propagation. For example, suppose a rectangle is being cut in half by a vertical line into two rectangles U and D. The x-length of a net involved in terminal propagation will be at least the minimum length of the new rectangles U and D if this net is cut, since this net will have to cross the whole length of either U or D. One-sided and two-sided constructions of the initial partition are used so that most of the nets involved in terminal propagation are not cut by the initial partition and are thus favored to remain uncut in later stages of the partitioning algorithm.

6. TUNING OF ROW-PLACE

In this section we present some enhancements used in the implementation of ROW-PLACE.

Preprocessing: The initial rectangulation consists of one slab that spans all the rows and containing one rectangle R. Let l and h be the length and height of R. Normally, R is a square, i.e., $l = h$. However, when $N = [l/h] > 1$, better performance were achieved by first cutting R by vertical lines into N almost-square rectangles. This was done by repeatedly cutting the longest rectangle in the slab into two rectangles by a vertical line, where the target length of the leftmost of the two new rectangles was set equal to h. This process terminates when the length of the longest rectangle in the slab is strictly less than $1.75h$.

Row length adjustment: During y-refinement, each slab that spans more than one row is cut into two slabs by a horizontal line. Consider a slab S consisting of k rectangles $R_1,\ldots,R_k$ listed in their order from left to right. The process of dividing S into two slabs proceeds by dividing each of rectangle R_i of S into two rectangles U_i and D_i, where U_i and D_i belong to the upper and lower new slabs respectively. Ideally, U_i and D_i should have an equal length. But in practice this may not be achieved all the time. If no adjustments are made, the lengths of the final rows of the placement may be unbalanced. In addition, unbalanced cuts of earlier (leftmost) rectangles in a slab causes opposite shifts in the position of later rectangles in the two new slabs. These opposite shifts cause nets to stretch in the x-direction and thus increase the

wire-length. This problem is corrected by processing the rectangles $R_1,\ldots,R_k$ in that order. When R_i is being cut into U_i and D_i, the previous rectangles $R_1,\ldots,R_{i-1}$ have already been cut. Let *top* (*bottom*) denote the total length of $U_1,\ldots,U_{i-1}$ ($D_1,\ldots,D_{i-1}$). The target length of U_i is set equal to the length of R_i plus *bottom-top* to adjust for previous length imbalance.

Another effort to balance the length of rows was made by restricting x-refinements. Specifically during x-refinement, a rectangle that spans more than one row is not allowed to be cut unless the number of nodes in it is greater than $UB = min\ (16,[high/low])$[1] where *low* and *high* are the minimum and the maximum node length. This is done so that in subsequent y-refinements, the number of nodes per rectangle is large enough to permit the partitioning algorithm to achieve the desired size ratio of the two parts. For example, suppose the first rectangle in a slab has two nodes of lengths 10 and 200, respectively. During y-refinement, such rectangle is problematic because it cannot be cleanly divided into two rectangles of about equal length and thus it will lead to a large size imbalance that decreases the placement quality. The above restriction on x-refinements is meant to avoid the appearance of such problematic rectangles.

Iteration of refinements: Due to the use of terminal propagation, partitioning of the nodes of one rectangle is influenced by the coordinate of nodes in other rectangles. Thus it is possible to get better performance by iterating x-refinements and y-refinements as long as the wire-length of the current placement can be improved. Normally each refinement is iterated 3 to 4 times.

Iteration of the partitioning algorithm: When the input subcircuit to the partitioning algorithm does not contain any of the two bias nodes, the initial partition is randomly generated. To improve performance in this case, the best partition generated by *LIMIT* runs of the partitioning algorithm is used, where *LIMIT* is a user-chosen parameter. In our experimentation, *LIMIT* was set equal to 5.

Local improvements: Two-interchange of nodes is used as a last step in improving the wire-length. However, nodes are only allowed to move locally. More precisely, the nodes are stored in a 2-dimensional table T. A node in $T(i, j)$ can only be interchanged with a node in $T(m, n)$, where $|i - m| \leq$ *ROW_RANGE* and $|j - n| \leq$ *RANGE*. *ROW_RANGE* and *RANGE* are two user-specified parameters, and they were respectively set to 1 and 5 in our experimentation.

After one two-interchange step, a node in a row may overlap with other nodes in the same row. Node overlaps are removed by adjusting node positions by a sweep of each row from left to right.

7. EXPERIMENTAL RESULTS

All our experiments were performed on a DEC 5000-240 workstation with 96 Megabytes RAM. The results of ROW-PLACE were compared with those of Timberwolf 6. Due to the use of randomness, both ROW-PLACE and Timberwolf 6 produce different results in different runs. For this reason, all results used are averages of 5 different runs in each case.

The circuits used are listed in Table I in increasing

TABLE I The circuits.

ckt	node	net	row
good	60	60	6
fract	125	128	6
t200g	200	300	7
ckta7	469	451	4
primary1	752	829	17
ckt5	800	684	8
t1000g	1000	1000	9
struct	1888	1888	20
ckta	2357	2167	15
primary2	2907	2961	23
test2	2976	3027	19
test1	3060	3123	21
biomed	6417	5711	44
industry2	12142	12949	69
industry3	15059	21807	52

order of size. The circuits Primary 1 and Primary 2 are two benchmark circuits from the 1987 Physical Design Workshop. The circuits fract, struct, biomed, industry2, industry3 are benchmark circuits from the 1991 Physical Design Workshop.

The first set of experiments shows the improved performance of ROW-PLACE due to preprocessing for those circuits with aspect ratio different from 1. In Table II, the wire-length results of ROW-PLACE without preprocessing (W_{np}) are expressed as percentages over the wire-length produced by ROW-PLACE with preprocessing (W_p), i.e., the numbers shown are of the form $(W_{np} - W_p) / W_p \times 100$. The number shown in the bottom corner of Table II is $(\Sigma W_{np} - \Sigma W_p) / \Sigma W_p \times 100$. For individual circuits, the improvement due to preprocessing ranges from 2.9% for circuit t200g to 22.9% for circuit ckta7. The overall improvement for all circuits is 9%.

The second set of experiments is intended to show the contribution of some specific parts of ROW-PLACE: (1) improvement due to local node interchange, (2) improvement due to the use of one-sided and two-sided constructions to generate the initial partition of the partitioning algorithm, and (3) the improvement due to the use of BISECT as the partitioning algorithm. The results are shown in Table III and are expressed as wire-length percentages over the wire-length produced by ROW-PLACE. The first column (local) shows the improvement due to the local node interchange step at the end of ROW-PLACE. The second column (random) shows the improvement due to the use of one-sided and two-sided constructions to generate the initial partition of BISECT over the use of a random initial partition. The third column (fm) shows the improvement due to the use of BISECT rather than the Fiduccia-Mattheyses algorithm (FM) [27] as the partitioning algorithm in ROW-PLACE. The last column (fm_random) expresses the results of ROW-PLACE using FM with a random initial partition as the partitioning algorithm, as percentages over the results of ROW-PLACE. The bottom row (total) of Table III expresses the results of other algorithms as percentages over the results of ROW-PLACE in terms of the sum of the wire-length for all the circuits involved. The following observation can be made:

1) The improvement due to local node interchange ranges from 1.5% for the circuit industry3 to 7.6% for the circuit good. The overall improvement for all circuits is 2.2%.
2) The overall improvement for all circuits due to the use of one-sided and two-sided constructions of the initial partition over using of a random initial partition in BISECT is 2.4%. However, ROW-PLACE using random initial partition in BISECT generated better results for some individual circuits as indicated by the negative number in the second column of Table III. This is mostly due to

TABLE II Preprocessing improvement.

ckt	l/h	percent
t200g	2	2.9
ckta7	8	22.9
ckt5	6	10.2
t1000g	6	14.6
ckta	2	5.2
test2	4	19.0
test1	2	3.3
total		9.0

TABLE III Effect of specific feature of ROW-PLACE.

ckt	local	random	fm	fm_random
good	7.6	0.4	0.4	−0.5
fract	6.8	−2.9	0.3	1.2
t200g	3.9	0.1	1.9	1.0
ckta7	5.4	3.7	17.0	23.6
primary1	4.9	0.4	1.3	6.6
ckt5	3.5	−0.1	8.4	9.7
t1000g	6.6	−1.8	−2.8	1.9
struct	4.9	−3.7	16.9	21.4
ckta	3.8	−0.4	15.9	28.4
primary2	2.6	0.6	8.1	13.8
test2	3.3	−0.2	15.1	20.4
test1	4.9	2.2	10.2	26.6
biomed	4.2	2.4	7.3	24.0
industry2	3.0	2.2	16.8	45.5
industry3	1.5	3.0	7.2	16.6
total	2.2	2.4	9.4	22.5

the stability of BISECT as is demonstrated in [25,26]. Nevertheless, one-sided and two-sided constructions of the initial partition lead to better results overall and for most individual circuits.

3) The third column in Table III shows the advantage of using BISECT rather than FM as the partitioning algorithm. Except for the circuit t1000g, ROW-PLACE performed better using BISECT. The improvement can be as much as 17% for the circuit ckta7 and is 9.4% overall.
4) The 4th column in Table III shows the significance of using one-sided and two-sided constructions to generate the initial partition for an unstable algorithm such as FM. By using FM with random initial partition instead of BISECT in ROW-PLACE, the results are 22.5% worse in comparison with 9.4% worse when one-sided and two-sided constructions are used to generate the initial partition of FM. This is to be contrasted with the 2.4% deterioration in performance when BISECT with random initial partition was used in ROW-PLACE (the second column). These results show that one-sided and two-sided constructions provide good starts for iterative improvement, while at they same time they show that BISECT is not as sensitive as FM to the initial partition as is indicated in [25,26].

The third set of experiments is intended to show the good performance of ROW-PLACE in comparison with Timberwolf 6 [7], a simulated-annealing based algorithm and widely recognized as the champion of row-based placement algorithms. The default parameter setting were used to run Timberwolf 6. Timberwolf 6 was also used with the parameter TWSCfast set to 10 (tw_fast(10)). This effectively speeds Timberwolf 6 by a factor of 10 so that its running time is comparable to ROW-PLACE. The purpose of this is to determine whether or not Timberwolf 6 is capable of producing good placements when it is run at the same speed as ROW-PLACE. ROW-PLACE was run in three different settings: (1) the usual setting as described in the paper (usual), (2) the usual setting without iterating refinements (no_iter), and (3) the same as the no_iter setting without iterating the partitioning algorithm (*LIMIT* = 1) and with limiting node interchange to adjacent nodes in the same row (fast).

The wire-length results are compared in Table IV. The results of each algorithm are expressed as percentages over the results of Timberwolf 6. The bottom line of Table IV shows percentages over the results of Timberwolf 6 in terms of the sum of the wire-length over all circuits.

The timing results are shown in Table V and they are expressed as multiples of the run-time of ROW-PLACE using the third setting (fast), which is shown in CPU seconds in the first column of this table.

The results of tw_fast(10) are significantly worse than those of Timberwolf 6 and are in fact the worst. The results of ROW-PLACE(usual) range from 7.8% better for the circuit biomed to 7.1% worse for the circuit primary2, and are overall within 1.2% of the results of Timberwolf 6. The results of ROW-PLACE(no_iter) range from 5.1% better for the circuit biomed to 8.8% worse for the circuit primary2, and are overall within 5.4% of the results of Timberwolf 6. The results of ROW-PLACE(fast) are always worse than Timberwolf 6 but are consistently better than tw_fast(10) except for the small circuits good and fract. Overall The results of ROW-PLACE(fast) are 16.3% worse than the results of Timberwolf 6 and are about 18.4% (= 34.7% − 16.3%) better than tw-

TABLE IV Wire-length comparisons.

ckt	tw_fast(10)	usual	no_iter	fast
good	3.5	0.0	−0.4	5.7
fract	8.2	3.6	3.6	15.2
t200g	5.8	−1.1	−1.0	3.8
ckta7	43.8	−0.8	7.5	16.3
primary1	31.7	4.2	6.4	15.4
ckt5	39.2	3.1	6.7	16.9
t1000g	9.0	−1.7	−3.0	1.5
struct	43.3	−3.6	−2.5	12.2
ckta	34.4	2.3	6.4	18.9
primary2	29.2	7.1	8.8	22.5
test2	46.5	0.6	7.8	12.6
test1	57.1	0.2	3.5	13.9
biomed	71.4	−7.8	−5.1	11.0
industry2	42.9	1.3	3.3	17.7
industry3	29.4	1.5	6.8	16.0
total	34.7	1.2	5.4	16.3

TABLE V Time comparisons.

ckt	fast(s)	usual	no_iter	tw_fast(10)	tw
good	0.3	8.1	6.3	142.4	1418.8
fract	0.54	8.4	6.0	64.5	607.4
t200g	2.78	8.4	6.7	40.9	385.2
ckta7	3.24	6.9	4.6	12.7	136.1
primary1	6.6	7.6	4.3	10.5	120.2
ckt5	6.36	7.7	4.3	10.5	131.8
t100g	22.38	11.8	7.5	13.6	129.1
struct	15.26	8.8	4.9	14.0	162.5
ckta	20.94	10.0	4.8	18.3	173.0
primary2	33.12	11.6	6.0	15.7	160.3
test2	32.52	9.1	5.1	16.0	171.0
test1	34.42	9.4	5.1	14.8	154.5
biomed	81.46	11.0	5.9	15.6	174.5
industry2	262.6	13.4	5.8	15.4	152.0
industry3	371.62	17.1	7.9	16.0	170.2
total	894.14	14.0	6.6	15.8	163.8

_fast(10). The timing results in Table V show that ROW-PLACE(usual), ROW-PLACE(no_iter), and ROW-PLACE(fast) are respectively 11.7 = 163.8/14.0, 24.8 = 163.8/6.6, and 163.8 times faster than Timberwolf 6. Table V also shows that tw_fast(10) is slower than ROW-PLACE(usual), while Table IV shows that the results of tw_fast(10) are significantly worse than the results of ROW-PLACE(usual). This shows that Timberwolf 6 cannot achieve comparable results to those of ROW-PLACE in the same amount of running time.

8. CONCLUDING REMARKS

In this paper, a fast and effective algorithm, ROW-PLACE, has been presented. ROW-PLACE achieves comparable results to those of Timberwolf 6 in 12 times faster running time. Also, good placements can be achieved in 25 times faster than Timberwolf 6 using ROW-PLACE(no_iter). The good results of ROW-PLACE are achieved using an improved clustering-based partitioning algorithm in combination with constructive methods that reduce the wirelength of nets involved in terminal propagation. This shows that if a good partitioning algorithm is used, min-cut placement can achieve simulated-annealing quality placement in much less time. ROW-PLACE is quite fast. For the circuit industry3 with 15059 nodes and 21807 nets, ROW-PLACE generated a placement in less than two hours (6355 seconds). Therefore, ROW-PLACE is suitable for use on large problems.

Acknowledgements

I am indebted to Dr. Carl Sechen for providing me with a copy of Timberwolf 6 and I wish to express to him my sincere thanks. This work was supported by the National Science Foundation under Grant MIP-9208293.

References

[1] B. Preas and P. Karger, "Automatic Placement: A Review of Current Techniques," *Proceedings of the 23st Design Automation Conference*, pp. 622–629, 1986.

[2] S. Goto and T. Matsuda, "Partitioning, Assignment and Placement," *Layout Design and Verification, T. Ohtsuki, Ed.* New York, NY: North-Holland, 1986.

[3] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, New York, NY: John Wiley & Sons, 1990.

[4] K. Shahookar and P. Mazumder, "VLSI Cell Placement Techniques," *Acm Computing Surveys*, vol. 23, no. 2, June 1991.

[5] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, pp. 671–680, May 1983.

[6] M. A. Breuer, "Min-Cut Placement," *Journal of Design Automation and Fault-Tolerant Computing*, vol. 1, no. 4, pp. 343–362, October 1977.

[7] C. Sechen and A. Sangiovanni-Vincentelli, "The Timberwolf Placement and Routing Package," *IEEE Journal of Solid-State Circuits*, vol. SC-20, pp. 510–522, April 1985.

[8] K. Loosemore, "Automated Layout of Integrated Circuits," *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 665–668, 1979.

[9] W. Dai and E. Kuh, "Simultaneous Floor Planning and Global Routing for Hierarchical Building-Block Layout," *IEEE Transactions on Computer-Aided Design*, vol. CAD-6, pp. 828–837, September 1987.

[10] M. Burstein, "A Non Placement/Routing Approach to Automation of VLSI Layout Design," *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 756–759, 1982.

[11] P. Suaris and G. Kedem, "An Quadrisection-Based Combined Place and Route Scheme for Standard Cells," *IEEE Transactions on Computer-Aided Design*, vol. 8, no. 3, pp. 234–244, March 1989.

[12] R. M. Kling and P. Banerjee, "ESP: Placement by Simulated Evolution," *IEEE Transactions on Computer-Aided Design*, vol. 8, no. 3, pp. 245–256, March 1989.

[13] J. Cohoon and W. Paris, "Genetic Placement," *Proceedings*

of the International Conference on Computer-Aided Design, pp. 422–425, 1986.

[14] K. Shahookar and P. Mazumder, "A Genetic Approach to Standard Cell Placement Using Meta-Genetic Parameter Optimization," *IEEE Transactions on Computer-Aided Design*, vol. 9, no. 5, pp. 500–511, May 1990.

[15] M. Hanan and J. Kurtzberg, "Placement Techniques," *Design Automation of Digital Systems, Theory and Techniques, vol. 1, M. Breuer*, Ed. Englewood Cliffs, NJ: Prentice-Hall, 1972.

[16] J. Cohoon and S. Sahni, "Heuristics for the Board Permutation Problem," *Proc. Int. Conf. Computer-Aided Design*, pp. 81–83, 1983.

[17] D. Schuler and E. Ulrich, "Clustering and Linear Placement," Proc. 9th Design Automation Conference, pp. 57–62, 1972.

[18] S. Chowdhury, "Analytical Approaches to the Combinatorial Optimization in Linear Placement," *IEEE Trans. Computer-Aided Design*, vol. 8, no. 6, pp. 630–639, June 1989.

[19] J. P. Blanks, "Near-optimal Placement Using a Quadratic Objective Function," *Proceedings of the 21st Design Automation Conference*, pp. 602–615, June 1985.

[20] A. E. Dunlop and B. W. Kernighan, "A Procedure for Placement of Standard-Cell VLSI Circuits," *IEEE Transactions on Computer-Aided Design*, vol. CAD-4, no. 1, pp. 92–98, January 1985.

[21] K. Antreich, F. Johannes, and F. Kirsch, "A New Approach for Solving the Placement Problem Using Force Models," *Proc. IEEE International Symposium on Circuits and Systems*, pp. 481–486, 1982.

[22] K. Hall, "An r-dimensional Quadratic Placement Algorithm," *Management Science*, vol. 17, no. 3, pp. 219–229, 1970.

[23] J. Frankle and R. Karp, "Circuit Placement and Cost Bounds by Eigenvector Decomposition," *Proceedings of the International Conference on Computer-Aided Design*, pp. 414–417, 1986.

[24] C. K. Cheng and E. S. Kuh, "Module Placement Based on Resistive Network Optimization," *IEEE Transactions on Computer-Aided Design*, March 1984.

[25] Y. Saab, "Post-Analysis-Based Clustering Dramatically Improves the Fiduccia-Mattheyses Algorithm," *European Design Automation Conference*, Hamburg, Germany, pp. 22–27, September 1993.

[26] Y. Saab, A Fast and Robust Network Bisection Algorithm," *IEEE Trans. Computers*, vol. 44, no. 7, pp 903–913, July 1995.

[27] C. Fiduccia and R. Mattheyses, "A Linear-Time Heuristics for Improving Network Partitions," *Proceedings of the 19th Design Automation Conference*, pp. 175–181, January 1982.

Endnote

1. The value of *UB* was restricted to be no more than 16 because the ratio *high/low* can be very large for some circuits.

Author Biography

Youssef G. Saab received the B.S. degree in computer engineering and the M.S. and the Ph.D. degrees in electrical engineering from the university of Illinois at Urbana-Champaign, Urbana, IL, in 1986, 1988, and 1990, respectively. He is currently an assistant professor in the department of computer engineering and computer science at the university of Missouri-Columbia.

From January 1986 to July 1990, he was a research assistant at the Coordinated Science Laboratory, Urbana, IL. His research interests include computer-aided design and layout of VLSI circuits, combinatorial optimization, computational geometry, and graph algorithms.

Dr. Saab is a member of IEEE, ACM, Tau Beta Pi, Eta Kappa Nu, Phi Kappa Phi, and the Golden Key National Honor Society.