Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

A NEW PLACEMENT ALGORITHM FOR VERY LARGE-SCALE IC DESIGNS

Jason Cong
Antonios Papandreou

October 1992
CSD-920054

A New Placement Algorithm for Very Large-Scale IC Designs

Jason Cong and Antonios Papandreou
Department of Computer Science
University of California, Los Angeles
Los Angeles, CA 90024

Abstract

Placement is the most important yet the most difficult step in VLSI layout design. The simulated annealing based placement algorithms have been widely used in practice. The best known algorithm in this class is TimberWolf, which produces high quality placement solutions compared to other existing methods at the expense of long computation time. Little progress has been made in the past a few years in developing new placement methods which can consistently outperform TimberWolf by a significant margin in both runtime and the quality of the solution. In this paper, we present a new placement method for large-scale cell-based IC designs. Compared with TimberWolf6.0 on the MCNC benchmarks of sizes ranging from a few hundred cells to over 12,000 cells, on average our placement algorithm reduces the runtime by a factor of 10, reduces the total wirelength by 37%, reduces the total number of tracks by 14%, and reduces the total number of feedthroughs by a factor of 155. Our algorithm successfully reduces the two-dimensional placement problem into a sequence of one-dimensional placement problems so that each of them can be solved independently using the rectilinear distance facility location formulation. The complexity of our algorithm is $O(n \log n)$ (where n is the number of cells in the design), which makes it very efficient for large-scale IC designs.

1. Introduction

Due to its inherent complexity, the VLSI layout design process is divided in general into two steps: placement and routing. The goal of the placement step is to map the circuit components onto positions on a layout surface. The objective of the routing step is to connect these placed components properly according to the netlist specification.

Placement is the most crucial step in the layout design process since the placement result affects significantly the quality of the subsequent routing solution. Due to its theoretical and practical importance, the placement problem has been studied extensively in the past two decades. The existing placement algorithms can be divided in two major categories: constructive placement or iterative placement. *Constructive placement algorithms* start with an unplaced netlist and construct a complete placement. The algorithms in this class include the cluster growth algorithms [Ku65, HaKu72, ScUl72, CoCa80, Ka83, OdIW85, OdHI87], the partitioning-based algorithms [Br77, StKK79, KaCK82, KoTI83, DuK85, LaDi86, GaVL91, GaVL92], the analytical placement algorithms (based on linear assignment [Ak81, JaKu89, SiDJ91, Yak92], quadratic or convex optimization [Ha70, HaKu72, KlSJ91, SrCK91, GaVL91, GaVL92], or network flows [DoJS92]), and the branch-and-bound algorithms [Gi62, HaKu72]. On the other hand, iterative placement algorithms start with a given complete placement and go through a number of local refinement steps to obtain an improved placement solution. The algorithms in the class include the pairwise interchange methods [HaWA76, Sc76, KhP77, IoKB83], the force directed methods [HaWA76, QuBr79, Go81] , and the simulated annealing based methods [KiGV83, SeS84, SeLe87]. More detailed survey of the existing placement algorithms can be found in [PrLo88] and [Le90]. Among these methods, the simulated annealing algorithm has been used widely in practice. The best known algorithm in this class is TimberWolf, which is the core of many commercial and in-house placement tools used in industry. Despite its long runtime, the TimberWolf package consistently produces high quality placement solutions compared to other existing placement methods. Many attempts have been made in the past few years in searching for more effective placement methods, but little progress has made in terms of improving the quality of the TimberWolf placement solutions (although some recent placement algorithms can reduce the runtime considerably). In the most recent placement contest (named Hinting Timberwolves) at the 1992 MCNC International Layout Synthesis Workshop, the Gordian/Domino placement package [KlSJ91, DoJS92] challenged TimberWolf. Gordian uses the linearly constrained quadratic programming formulation for global optimization, and Domino is a recently developed iterative improvement procedure based on the network flow formulation which is used to further refine the placement solution by Gordian. TimberWolf and Gordian/Domino were compared on a real design of 13770 cells and 16642 nets (the example was not revealed to the two groups before the contest so that problem specific tuning was not possible). Although Gordian/Domino ran 5 times faster than TimberWolf (2.9 hours versus 14.3 hours), the quality of the two placement solutions were almost identical (the difference in the final layout area is less than 1%). This result and other experimental results lead to an interesting question, that is, if the placement solution produced by the simulated annealing algorithm is indeed very close to optimal. On the other hand, rapid advances in VLSI

technology has increased the packing density by a factor of 10 to 100 in the past decade. The drastic increase in design complexity has made the simulated annealing based methods unacceptable due to their extremely long runtime. The technological advance poses another pressing and challenging question to the layout community: Is it possible to develop a much faster placement algorithm which can produce equivalent or even better results than the simulated annealing algorithm?

These questions motivate our research on fast placement algorithms for very large-scale IC designs. Our answers to these questions are surprising: we have developed a new placement algorithm, named AKROPOLIS, which consistently outperforms TimberWolf in terms of both runtime and the quality of the placement solution by a significant margin on the MCNC layout benchmarks of sizes ranging from a few hundred cells to over 12,000 cells. On average, AKROPOLIS reduces the runtime by a factor of 10, reduces the total wirelength by a 37%, reduces the total number of tracks by 14%, and reduces the total number of feedthroughs by a factor of 155 compared to TimberWolf6.0. Our algorithm is a combination of the partitioning-based technique and the analytical placement technique. It successfully reduces the two-dimensional placement problem into a sequence of one-dimensional placement problems so that each of them can be solved independently using the rectilinear distance facility location formulation. The complexity of our algorithm is $O(n \log n)$ (where n is the number of cells in the design), which makes it very efficient for very large-scale IC layout designs.

As the VLSI fabrication technology reaches submicron device dimension and gigahertz frequency, interconnection delay has become the dominant factor in determining system performance. This leads to recent interests in performance-driven placement in which main objective is to minimize the interconnection delay. The performance-driven placement algorithms can be broadly grouped in two categories: net-based algorithms [Do90, SuSh90, LiDu90, GaVL92] and path-based algorithms [SrCK91, JaKu89, MaLi89, HaNY87]. In our formulation, nets can be assigned variable weights determined by critical path analysis, and timing-critical nets can be optimized with higher priorities by our placement algorithm. Since AKROPOLIS obtains drastic reduction in total wirelength (up to 54%), we expect that its placement solution has a much shorter interconnection delay in general as well.

The remainder of this paper is organized as follows: Section 2 presents the formulation of the problem. Section 3 discusses our placement algorithm in detail. The experimental results and comparative study are presented in Section 4. Finally, conclusions and future extensions are presented in Section 5.

2. Formulation of the Problem

A circuit consists of a set of components and a set of primary input/output (I/O) pads. Interconnections among the components and the I/O pads are specified by a netlist, in which each net specifies a set of pins that have to be electrically connected. The goal of placement is to compute a mapping of the components in a circuit onto positions on a layout surface while optimizing certain objective functions.

Several objective functions have been proposed. Commonly used objectives are the minimization of the total wirelength, the minimization of the layout area and the maximization of the routability of the

circuit. Performance-driven placement algorithms emphasize the minimization of interconnection delay to maximize the circuit speed.

The algorithm presented in this paper, called AKROPOLIS, solves the placement problem for row-based gate array and standard cell designs. In gate array designs, the components of the circuit are mapped onto a prefabricated, two-dimensional array of uncommitted simple gates. In standard cell designs, each component (cell) is from a predesigned finite cell library. The cells have approximately the same height, but their width may vary considerably. In both gate-array and standard cell designs, the components are rectangular in shape and are to be placed in rows of roughly equal length. The I/O pads are on the periphery of the chip. In our formulation we assume that the I/O pad positions are preassigned.[1] The connections between the cells are realized by metal wires in the rectangular routing areas between the cell rows (called channels). An example of a standard cell design is shown in Figure 2.1. The objective function used in AKROPOLIS is the minimization of the total weighted wirelength $\sum_{i \in \Pi} w_i l_i$, where l_i is the wirelength of net i, computed by the half perimeter of the minimum bounding box of the net, w_i is the weight of net i, indicating the criticality of the net, and Π is the set of nets in the design.

3. The AKROPOLIS Placement Algorithm

Most partitioning-based algorithms reduce a two-dimensional placement problem into a set of two-dimensional placement problems of smaller size by repeatedly bipartitioning the circuit in alternating directions. The main contribution of AKROPOLIS is the transformation of the two-dimensional placement problem into a sequence of one-dimensional placement problems which can be solved independently. The input of AKROPOLIS consists of a netlist and cell descriptions, the I/O pad positions

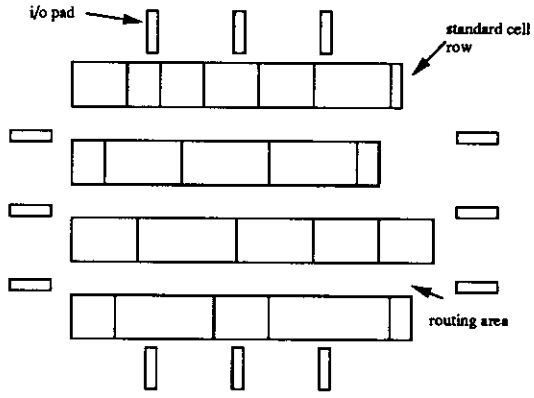

Figure 2.1: A standard cell design

[1] If I/O pads are not placed, our algorithm uses the TimberWolfMC to determine the I/O pad positions. In Section 5 we shall discuss how to extend our algorithm to handle the case when the I/O pad positions are not given.

and the number of cell rows in the design (determined according to the given chip aspect ratio). The output specifies the exact positions of all the circuit cells. Our algorithm consists of two phases :

- Phase 1: Placement of cells into rows
- Phase 2: Placement of cells in each row

Phase 1 assigns the cells to the rows in the design. Phase 2 processes the rows of the design one by one and determines the exact position of each cell inside the row.

In the following two sections, the details of the two phases are presented.

3.1. Phase 1: Placement of Cells into Rows

Phase 1 is relatively simple. During this phase, we assign each cell to one of the cell rows in the design, based on recursive bipartitioning of the cells in the design. The bipartitioning algorithm is based on a modification of the min-cut heuristic[2] procedure of Fiduccia and Mattheyses (the FM algorithm) [FiMa82]. The goal of the FM algorithm is to partition the components into two blocks of approximately equal size while minimizing the number of connections (net cut) between them. The FM algorithm goes through a number of *passes* to improve an initial partitioning. The time complexity of each pass of the FM algorithm was shown to be linear in terms of the total number of pins in the design. Experimental results showed that a very small number of passes is required in practice (2-5 passes) [FiMa82]. In our application of the FM algorithm, we added the following two simple enhancements:

(1) The capability of specifying "fixed" cells. The assignment of the fixed cells to the two blocks is specified by the user. This can be easily accomplished by "locking" the fixed cells before each pass.

(2) We introduce two parameters r and s to specify the balancing condition as follows :

$$(|A|+|B|)\cdot(r-s)<|A|<(|A|+|B|)\cdot(r+s)$$

where r is the desired fraction of the total area of block A and s is a user specified deviation percentage (e.g. if we set r=60% and s=5% the balancing condition is met if the area of block A is within the range of 55% to 65% of the total area).

Assume that there are M rows in the design. We number these rows 1,2,..,M starting with the bottom row. During this phase all the cells of the circuit are recursively bipartitioned until each cell belongs to only one row. Initially, each cell can be assigned to any of the rows in the design. After each bipartition, the range of permissible rows for each cell decreases by a factor of two. The process stops when each cell is assigned to a unique row.

During each recursion, we distinguish three classes of cells.

(1) The set of cells U that are going to be partitioned in this level, with the permissible row range [*low_row* , *high_row*] (*low_row* <*high_row*).

---

[2] The bipartitioning problem is proved to be NP-complete

(2) The cells that have already been confined to rows $[1, low\_row)$ $(low\_row > 1)$.
(3) The cells that have already been confined to rows $(high\_row, M]$ $(high\_row < M)$.

Let $cells(a)$ denote the set of cells connected by net a. We collapse all cells in class 2 (if any) together with all the bottom I/O pads into a supercell called the *source*. For any net a, the source belongs to $cells(a)$ if any cell in the source belongs to $cells(a)$. We consider the source occupies zero area. Similarly, all cells in class 3 (if any) together with all the top I/O pads are collapsed into a supercell called the *sink*. For any net a, the sink belongs to $cells(a)$ if any cell in the sink belongs to $cells(a)$. The sink is considered to occupy zero area as well. For the first recursion, the source contains only the bottom I/O pads and the sink only the top I/O pads. It is clear that the addition of the two supercells to the set of cells being partitioned provides global interconnection information, which improves the quality of the final partition. (Note that the addition of supercells is similar to the terminal propagation technique discussed in [DuK85]. However, since we are solving a one-dimensional placement problem in this phase, our solution is much simpler.) We partition the set of cells U together with the source and the sink into two blocks A and B using the modified FM algorithm according to the following specifications:

(1) Block A has to contain the source.
(2) Block B has to contain the sink.
(3) Let $mid\_row = \left\lfloor \frac{low\_row + high\_row}{2} \right\rfloor$. The desired area ratio of block A is $r = \frac{mid\_row}{low\_row + high\_row}$. Moreover, the user can specify a parameter s, called *balance deviation*, which allows the final size of block A to differ from its targeted value by s percent.

After partitioning, cells in block A are confined to rows $[low\_row, mid\_row]$ and cells in block B are confined to rows $(mid\_row, high\_row]$. Because the partition computed by the modified FM algorithm is non-deterministic (the result depends on a random initial partition), we call the partitioning procedure $i1$ times at each step and use the best partitioning result, where $i1$ is a user specified parameter that affects both the running time of this phase and the quality of the result. In general, a larger value of $i1$ leads to better partitions but longer computational time. The parameter s affects the rowlength deviation. Smaller s leads to designs of more even rowlength.

At the end of phase 1 each cell has been assigned to one of the rows. The exact positions of the cells in each row are to be determined by the next phase.

3.2. Phase 2: Placement of Cells Inside Each Row

During this phase, the cells in each row are assigned to their exact locations. We process all the rows on a one-by-one basis starting from the bottom row. The placement of the cells in each row depends on the cell positions in the already placed rows underneath the current row and the I/O pad positions.

For each row, we have developed an analytical placement technique which determines the positions of all the cells in the row simultaneously. In particular, we formulate the single row placement problem as a generalization of the Rectilinear Distance Facility Location (RDFL) problem. In the next subsection we first describe the classical RDFL problem and the solution by Picard and Ratliff [PiRa78].

3.2.1. The Classical RDFL Problem and its Solution

The RDFL problem is a well studied problem in location theory. It is formulated as follows: Assuming that there are m old facilities located at coordinates (a_i,b_i) $(1\le i\le m)$ in the plane, determine the optimal locations of n new facilities in the plane such that the sum of weighted rectilinear distances between all facilities is minimized.

Formally, let w_{ij} denote the amount of interconnection between an old facility i and a new facility j $(1\le i\le m, 1\le j\le n)$ and u_{ij} denote the amount of interconnection between two new facilities i and j $(1\le i\le n, 1\le j\le n)$. Then, the RDFL problem is to find the coordinates (x_i,y_i) for each new facility i $(1\le i\le n)$ such that the function:

$$Z(X,Y)=\sum_{i=1}^{m}\sum_{j=1}^{n}w_{ji}(|x_j-a_i|+|y_j-b_i|)+(1/2)\sum_{i=1}^{n}\sum_{j=1}^{n}u_{ij}(|x_i-x_j|+|y_i-y_j|) \qquad \text{(E1)}$$

is minimized, where $X=(x_1,x_2,..,x_n)$ and $Y=(y_1,y_2,..,y_n)$. Note that $w_{ij}=w_{ji}$ and $u_{ij}=u_{ji}$ are implied in the formulation.

Since $Z(X,Y)$ is separable in terms of X and Y, it suffices to consider two one-dimensional RDFL problems independently:

$$minimize:\ G(X)=\sum_{i=1}^{m}\sum_{j=1}^{n}w_{ji}|x_j-a_i|+(1/2)\sum_{i=1}^{n}\sum_{j=1}^{n}u_{ij}|x_i-x_j| \qquad \text{(E2)}$$

$$minimize:\ H(Y)=\sum_{i=1}^{m}\sum_{j=1}^{n}w_{ji}|y_j-b_i|+(1/2)\sum_{i=1}^{n}\sum_{j=1}^{n}u_{ij}|y_i-y_j| \qquad \text{(E3)}$$

According to the results in [CaFS70] we know that there exists an optimum solution with $x_j\in\{a_1,..,a_m\}$ for each $j=1,..,n$. Moreover it is sufficient to consider the case where $a_{i+1}-a_i=1$ for $i=1,..,m-1$. Therefore, the one-dimensional RDFL problem can be simplified to the following form: Compute $X=\{x_1,..,x_n\}$ where $x_i\in\{1,..,m\}$ $(1\le i\le n)$ such that:

$$F(X)=\sum_{i=1}^{m}\sum_{j=1}^{n}w_{ji}|x_j-i|+(1/2)\sum_{i=1}^{n}\sum_{j=1}^{n}u_{ij}|x_i-x_j| \qquad \text{(E4)}$$

is minimum.

For each $i=1,..,m-1$, we denote the line segment from i to $i+1$ as segment i. Then, for any given X satisfying $x_i\in\{1,..,m\}$ $(j=1,..,n)$ we can decompose $F(X)$ into $m-1$ components, such that each component corresponds to the contribution to $F(X)$ by one of the $m-1$ line segments. Specifically, let $S_q=\{j\,|\,x_j\le q\}$ and $S'_q=\{j\,|\,x_j>q\}$. Then, the contribution of segment q, denoted as C_q, can be written as:

$$C_q(S_q,S'_q)=\sum_{j\in S_q}\sum_{i=q+1}^{m}w_{ji}+\sum_{j\in S'_q}\sum_{i=1}^{q}w_{ji}+\sum_{j\in S_q}\sum_{k\in S'_q}u_{jk} \qquad \text{(E5)}$$

and $F(X)$ can be written as :

$$F(X) = \sum_{q=1}^{m-1} C_q(S_q, S'_q) \qquad \textbf{(E6)}$$

For any given q, we say that the partition (S_q, S'_q) is optimal if $S_q \cup S'_q = \{1,..,n\}$, $S_q \cap S'_q = \emptyset$ and $C_q(S_q, S'_q)$ is minimum. A key result in [PiRa78] showed that for any q, if partition (S_q, S'_q) is optimal, then there exists an optimal solution $X^* = \{x_1^*,..,x_n^*\}$ to the one-dimensional RDFL problem such that $x_i^* \leq q$ if $i \in S_q$ and $x_i^* \geq q+1$ if $i \in S'_q$. Clearly, by recursively finding the optimal partition (S_q, S'_q) for each q, we can obtain an optimal solution to the one-dimensional RDFL problem. It was shown that the problem of computing an optimal partition (S_q, S'_q) for any q can be solved by finding a min-cut in a network.

For any q, we construct a network, called the *q-locale network*, as follows : there is one node for each new facility j (j=1,..,n) plus two nodes, called s (source) and t (sink). There are three types of undirected arcs in the network:

(1) (s, j) with capacities $c(s, j) = \sum_{i=1}^{q} w_{ij}$ for $j = 1,..,n$

(2) (t, j) with capacities $c(t, j) = \sum_{i=q+1}^{m} w_{ij}$ for $j = 1,..,n$

(3) (j, k) with capacities $c(j, k) = u_{jk}$ for $j = 1,..,n$ and $k = 1,..,n$ with $j \neq k$

Figure 3.1 shows a q-locale network with 3 new facilities and m old facilities.

If we partition the vertices $\{1,..,n\}$ of the q-locale network into two sets S_q and S'_q, then the vertex sets $\{s\} \cup S_q$ and $\{t\} \cup S'_q$ define a cut in the q-locale network. We denote this cut by (X_q, X'_q). Then, we have the following observations :

(1) The cost of the cut (X_q, X'_q) (i.e., the total capacity of the edges across the cut) in the q-locale network equals $C_q(S_q, S'_q)$ defined in Equation (E5).

(2) Finding an optimal partition (S_q, S'_q) is equivalent to finding a minimum cut in the q-locale network.

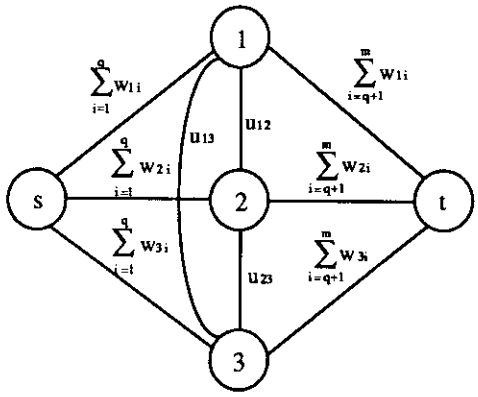

Figure 3.1: A q-locale network with m old and 3 new facilities

Since the min-cut problem in a network can be solved efficiently based on the max-flow computation [FoFu62], the optimal partition (S_q, S'_q) can be computed in polynomial time. For any segment q, after we compute the optimal partition (S_q, S'_q), the new facilities are partitioned into two sets: those located at positions less than or equal to q and those located at positions greater or equal to $q+1$. After we compute the optimal partition for each of the $m-1$ segments, each new facility is constrained to a unique location and a complete solution to the RDFL problem is obtained.

Let L_q denote the set of facilities restricted to the positions less than or equal to q, and G_q the set of facilities restricted to positions greater than or equal to q. The following algorithm computes an optimal solution to the RDFL problem :

Step 0. Set $N=\{1,..,n\}$, $L_m=G_1=N$, $L_i=\emptyset$ $(i=1,..,m-1)$, $G_i=\emptyset$ $(i=2,..,m)$ and $I=\{1,..,m-1\}$.
Step 1. If $I=\emptyset$ goto Step 5. Otherwise pick any q from I and let $I=I-\{q\}$.
Step 2. Let $L_q=\bigcup_{i\le q} L_i$ and $G_{q+1}=\bigcup_{i\ge q+1} G_i$. Consider the new facilities in L_q as old facilities located at q and the new facilities in G_{q+1} as old facilities located at $q+1$. Consider $N-\{L_q \cup G_{q+1}\}$ as the set of new facilities to be located. Construct the q-locale network for this problem.
Step 3. Find the minimum cut (S_q, S'_q) in the q-locale network defined in Step 2.
Step 4. Set $L_q=L_q \cup S_q$ and $G_{q+1}=G_{q+1} \cup S'_q$. Goto Step 1.
Step 5. For each $1\le i\le n$, if $i \in L_j \cap G_j$ then $x_i=j$.

The method requires the solution of at most $m-1$ min-cut problems on networks with at most $n+2$ vertices.

3.2.2. The Single-Row Placement Algorithm

The central part of Phase 2 of our algorithm is to solve the single-row placement problem. In order to place the cells in the current row we take into consideration the following three factors :

(1) The connections to all the already placed cells that are below the current row.
(2) The connections to all the I/O pads in the design.
(3) The connections among the cells in the current row.

The single-row placement algorithm consists of two stages. The first stage solves the generalized one-dimensional RDFL problem where the old facilities are the I/O pads and the cells that have already been placed, and the new facilities are the cells of the current row. Assume that there are n cells in the current row. If the placed cells and the I/O pads have m distinct x-coordinates $a_1<a_2<..<a_m$, according to the result in [CaFS70] we can assume that $a_i=i$ $(1\le i\le m)$. Then, the solution to the one-dimensional RDFL problem gives the x-coordinate of each cell in the current row $1\le x_j\le m$ $(1\le j\le n)$, which specifies the relative ordering of the cells in the current row. Without loss of generality, we assume that $m=2^k$ for some k in the following discussions. Note that overlap between cells in the current row may exist since the solution to the RDFL problem does not guarantee that each new facility occupies a distinct position.

The second stage of the single-row placement algorithm removes all the overlaps of the cells in the current row. That is, we decide the relative ordering of all new facilities placed at the same position, as determined in the first stage. Figure 3.2 illustrates the output of each of the two stages.

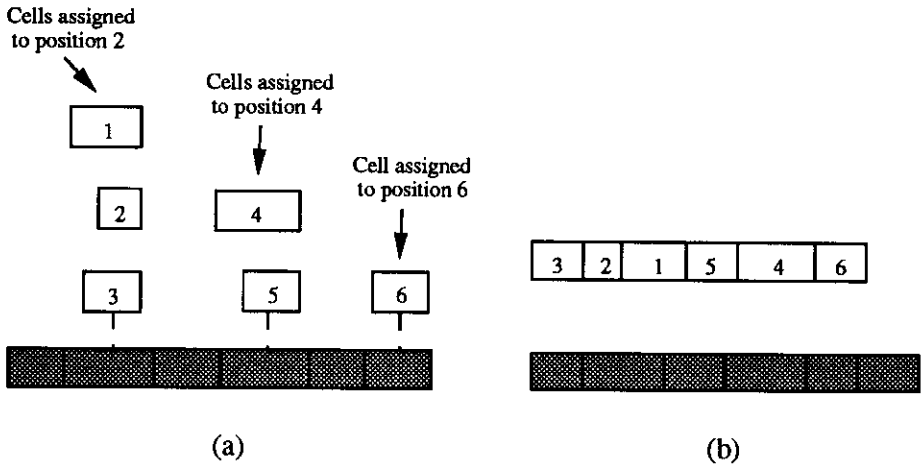

Figure 3.2: (a) Single-row placement based on the solution to the generalized RDFL problem. (b) Single-row placement after removing overlaps.

3.2.2.1. Stage 1: Solution to the Generalized RDFL Problem

Two difficulties arise when applying the classical RDFL problem to the single-row placement problem. First, the interconnection information of the circuit is usually a netlist (i.e. a hypergraph) instead of a simple graph, which makes the cost computation of a cut in the q-locale network complicated. Consider for example the q-locale network in Figure 3.3. If some net connects nodes 1,2,3 and t, the contribution of the net to the cost of the cut is just 1 instead of 3 as would be computed by adding the number of common connections between node t and each of the nodes 1,2 and 3.

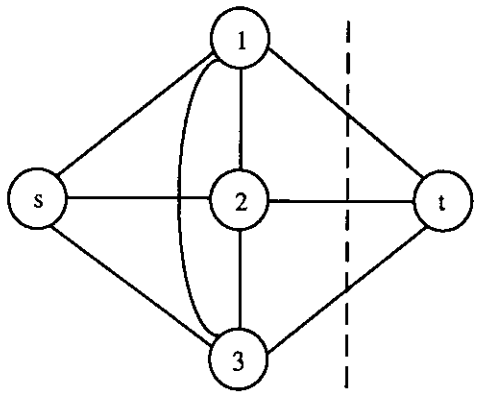

Figure 3.3: Computing the cost of a cut in the q-locale network

Second, based on the optimal solution to the one-dimensional RDFL problem, many cells in the current row may overlap. In our experiments, we observed that as many as 50% of the cells in the current row could occupy the same location. While in general overlaps are allowed during this stage, excessive overlap of the cells defies the purpose of determining relative cell ordering in the current row and is certainly not desirable.

In order to overcome these two difficulties, we modified the RDFL problem formulation to handle hypergraph interconnections and to limit overlap at each facility location. We call the resulting problem the *generalized RDFL problem*.[3] In our solution to the generalized RDFL problem, instead of finding a min-cut in each q-locale network, we find the minimum area-balanced cut in the *q-locale hypergraph* (to be defined below) using the modified FM partitioning method presented in Subsection 3.1. Although with this modification an optimal solution is no longer guaranteed, the application of the modified FM algorithm can easily handle interconnection information in the netlist form. Moreover, if we compute an area-balanced cut in each q-locale hypergraph, cells are evenly distributed over all the possible facility locations and the overlap is greatly reduced[4]. Our algorithm for solving the generalized RDFL problem is described as follows.

For each net a, let $h\_cells(a)$ be the subset of $cells(a)$ that contains the cells of net a that are assigned to the highest row (row with the highest index) below the current row. We assume that the global router will not introduce unnecessary feedthroughs when connecting cells in the same net. That is, if a net connects three cells in rows i, j and k with $i<j<k$, the global router will try to connect the cells in rows i and j and the cells in rows j and k, instead of connecting cells in rows i and k directly using feedthroughs. Therefore, when we place the cells in the current row, we consider only their connections to the cells in $\bigcup_{a\in\Pi'} h\_cells(a)$ and connections to the corresponding I/O pads, where Π' is the set of nets connecting to some cells in the current row.

During each iteration of our algorithm for solving the generalized RDFL problem, instead of constructing a q-locale network for each selected segment q, we construct the *q-locale hypergraph* defined as follows:

(1) The hypergraph contains one vertex for each new facility of this iteration, plus two vertices s (source) and t (sink).

(2) For each net a incident to some new facility, a hyperedge H_a is defined as follows:

$$H_a = \{cell \mid cell \in cells(a) \textbf{ and } cell \text{ is a new facility}\} \cup P \cup Q$$

where

[3]The classic RDFL formulation was also used by Marek-Sadowska and Lin [MaLi89]. They applied it directly to the two-dimensional placement problem. Moreover, they did not have overlap control in their formulation. Excessive overlaps were allowed and they were resolved afterwards using a procedure similar to the min-cut placement algorithm.

[4]Another advantage of using the modified FM algorithm is that it reduces the time complexity of the min-cut algorithm. The modified FM algorithm runs in linear time in practice while the min-cut algorithm requires cubic time for maximum flow computation.

$$P = \begin{cases} \{s\} & \text{if net } a \text{ contains an I/O pad assigned the left side of the chip or} \\ & h\_cells(a) \text{ contains an old facility whose } x\text{–coordinate} \le q \\ \varnothing & \text{otherwise} \end{cases}$$

and

$$Q = \begin{cases} \{t\} & \text{if net } a \text{ contains an I/O pad assigned the right side of the chip or} \\ & h\_cells(a) \text{ contains an old facility whose } x\text{–coordinate} \ge q+1 \\ \varnothing & \text{otherwise} \end{cases}$$

The weight of hyperedge H_a is set to be the same as the weight of net a. A *cut* in the q-locale hypergraph is a vertex partition (X, X') of the hypergraph such that $s \in X$ and $t \in X'$. The *cost of a cut* in the q-locale hypergraph is the sum of the weights of all the hyperedges across the cut. Then, we can generalize the results in [PiRa78] as follows.

Theorem 1: Given $X=(x_1,x_2,..,x_n)$, let $S_q=\{x_i \mid x_i \le q\}$ and $S'_q=\{x_i \mid x_i \ge q+1\}$. Then, X is an optimal solution to the RDFL problem with hyperedge connections if and only if (S_q, S'_q) is an optimal partition of the q-locale hypergraph for each q.

The proof of the theorem is similar to that in [PiRa78]. Based on this theorem, the generalized RDFL algorithm works as follows:

Step 0. Set $N=\{1,..,n\}$, $L_m=G_1=N$, $L_i=\varnothing$ $(i=1,..,m-1)$, $G_i=\varnothing$ $(i=2,..,m)$ and $I=\{1,..,m-1\}$.
Step 1. If $I=\varnothing$ goto Step 5. Otherwise pick q according to the binary search ordering (defined below) from I and let $I=I-\{q\}$.
Step 2. Let $L_q=\bigcup_{i \le q} L_i$ and $G_{q+1}=\bigcup_{i \ge q+1} G_i$. Consider the cells in L_q as old facilities located at q and the cells in G_{q+1} as old facilities located at $q+1$. Consider $N-\{L_q \cup G_{q+1}\}$ as the set of new cells to be located. Construct the q-locale hypergraph for this problem.
Step 3. Find the minimum area balanced cut (S_q, S'_q) in the q-locale hypergraph defined in Step 2, using the FM method with a user specified balance deviation s.
Step 4. Set $L_q=L_q \cup S_q$ and $G_{q+1}=G_{q+1} \cup S'_q$. Goto Step 1.
Step 5. For each $1 \le i \le n$, if $i \in L_j \cap G_j$ then $x_i=j$.

The order by which the segment q is chosen during Step 2 of the algorithm is according to the binary search ordering defined by the sequence: $\frac{m}{2}, \frac{m}{4}, \frac{3m}{4}, \frac{m}{8}, \frac{3m}{8}, \frac{5m}{8}, \frac{7m}{8}, \cdots$ which is the sequence that we follow to carry out binary search on interval $[1, m]$. Initially, the cells in the current row belong to the interval $[1, m]$. After $q=\frac{m}{2}$ is processed, each cell belongs to either interval $[1, \frac{m}{2}]$ or interval $[\frac{m}{2}+1, m]$. After $q=\frac{m}{4}$ is processed, each cell belongs to one of the intervals $[1, \frac{m}{4}]$, $[\frac{m}{4}+1, \frac{m}{2}]$, or $[\frac{m}{2}+1, 1]$. At each iteration, one interval is refined (being split into two smaller intervals of equal length). At any time, each cell in the current row belongs to one of the intervals. However, the

exact location of the cell in that interval is yet to be determined. This sequence was chosen because it enhances the even distribution of the cells in the current row over the m possible positions. In particular, we have the following result:

Theorem 2: Assuming that the cells of each row are of uniform length[5], the number of overlapping cells at each facility location in our solution to the generalized RDFL problem is no more than $\left\lceil (1+s)^{\log m} \cdot \frac{n}{m} \right\rceil$, where s is the balance deviation parameter used in the modified FM algorithm in Step 3.

Proof: Note that $\left\lceil \frac{n}{m} \right\rceil$ is a lower bound of the number of overlapping cells when we place n cells into m locations. Assume that $m=2^k$. We shall show by induction that during the execution of our algorithm, an interval of length 2^q $(q \le k)$ contains at most $\left\lceil \frac{(1+s)^{k-q} \cdot n}{2^{k-q}} \right\rceil$ cells.

For q=k it is trivial to see that the interval with length 2^k $(=m)$ contains n cells. Assume that the statement is true for q. We shall show that it is also true for $q-1$. The interval of length 2^{q-1} is obtained by partitioning an interval of length 2^q using the modified FM algorithm. If the balance deviation of this algorithm is s, according to the induction hypothesis, the interval of length 2^{q-1} contains at most

$$\left\lceil \frac{\frac{(1+s)^{k-q} \cdot n}{2^{k-q}}}{2} + s \cdot \frac{\frac{(1+s)^{k-q} \cdot n}{2^{k-q}}}{2} \right\rceil = \left\lceil \frac{(1+s)^{k-q+1} \cdot n}{2^{k-q+1}} \right\rceil = \left\lceil \frac{(1+s)^{k-(q-1)} \cdot n}{2^{k-(q-1)}} \right\rceil$$

cells.

Since this is true for any q, then for q=0 we obtain that the maximum number of cells at each old facility location (i.e. a unit length interval) is

$$\left\lceil \frac{(1+s)^k \cdot n}{2^k} \right\rceil = \left\lceil (1+s)^{\log m} \cdot \frac{n}{m} \right\rceil$$

□

Usually, m and n are roughly equal since Phase 1 tries to balance the row lengths. In this case, the maximum number of overlapping cells is bounded by $2 \cdot (1+s)^{\log m}$. It is clear that smaller s leads to fewer overlapping cells.

Upon completion of Stage 1, all cells in the current row have been assigned to one of the positions 1,..,m defined by the old facilities. This assignment specifies a partial order among the cells in the current row. The cell (or the group of cells) assigned to position i is (are) going to be placed to the left of the cell (or the group of cells) assigned to position j if $i<j$.

---

[5] Althought our algorithm handles variable cell lengths, this assumption is made for the sake of the simplicity of the analysis.

3.2.2.2. Stage 2: Overlap Removal

Each set of cells that are placed at the same position in the current row after Stage 1 forms a *group*. The objective of Stage 2 is to determine the relative ordering of the cells in each group. Let G_i denote the i-th cell group from the left in the current cell row. Moreover, let L_i represent the sum of cell lengths in G_i. Then, cells in G_1 will occupy the interval $[0, L_1]$. The cells in G_2 will occupy the interval $[L_1, L_1+L_2]$. In general, the cells in G_i will occupy the interval $[\sum_{k=1}^{i-1} L_k , \sum_{k=1}^{i} L_k]$.

For each group, the overlap removal problem is formulated as a maximum bipartite matching problem. If the cardinality of G_i is n_i, then the n_i cells of the group are to be matched to an equal number of positions. We assume that all the cells in G_i have the average cell length of G_i, denoted $\overline{l_i}$, and all the candidate positions for the cells in G_i are equally spaced in $[\sum_{k=1}^{i-1} L_k , \sum_{k=1}^{i} L_k]$. In addition, all the pins in each cell are assumed to be located in the middle of the cell.

Each vertex in one side of the bipartite graph represents a cell, and each vertex in the other side of the bipartite graph represents a position. The x-coordinate of the center of position r is:

$$X_r = \sum_{k=1}^{i-1} L_k + (r-\frac{1}{2})\cdot \overline{l_i}$$

The weight c_{jr} of the edge connecting cell j to position r in the bipartite graph represents the cost of assigning cell j to position r:

$$c_{jr} = (P_j^+ - P_j^-)\cdot | \sum_{k=1}^{i-1} L_k - X_r |$$

where P_j^+ is the number of nets whose minimum bounding box perimeter will increase when we move cell j from $\sum_{k=1}^{i-1} L_k$ to X_r and P_j^- is the number of nets whose minimum bounding box perimeter will decrease when we move cell j from $\sum_{k=1}^{i-1} L_k$ to X_r. Therefore, cost c_{jr} represents the increase of total wirelength when we shift cell j from $\sum_{k=1}^{i-1} L_k$ to X_r.

Let $\{x_{jr}\}$ be a set of 0/1 integer variables for $j=1,..,n_i$ and $r=1,..,n_i$. Solution $x_{jr}=1$ means that the matching includes the assignment of cell j to position r whereas $x_{jr}=0$ means that it does not. Clearly, we want to compute a complete bipartite matching such that $\sum_{j,k=1}^{n_i} (c_{jk} \cdot x_{jk})$ is minimized.

The minimum weighted complete bipartite matching problem can be efficiently solved using the algorithm presented in [PaSt82]. The solution of the bipartite matching problem defines the relative ordering of the cells inside each group. Since the relative order of the groups has been determined in Stage 1, after solving the minimum weighted complete bipartite matching problem for each group of cells during Stage 2, the relative order of all the cells in the current row has been decided and all the overlaps have been removed.

3.3. Overall Time Complexity

Let n be the number of cells in the design. We assume that the number of pins on each cell is bounded by a constant. In Phase 1, the modified FM partitioning algorithm is applied to 1 netlist of n cells, 2 netlists each of $n/2$ cells, 4 netlists each of $n/4$ cells, ..., and so on. Since each pass of the modified FM algorithm runs in linear time, the time complexity of Phase 1 is

$$k_{FM} \cdot O(n) + 2k_{FM} \cdot O(n/2) + 4k_{FM} \cdot O(n/4) + \cdots = k_{FM} \cdot O(n \log n)$$

where k_{FM} is the number of passes executed by the FM algorithm, which is usually a small constant (2-5) [FiMa82]. Without loss of generality, assume that there are r rows and that each row has n_0 cells after Phase 1. In Phase 2, for each single row placement problem, the modified FM algorithm is applied to 1 q-locale hypergraph with n_0 nodes, 2 q-locale hypergraphs each of $n_0/2$ nodes, 4 q-locale hypergraphs each of $n_0/4$ nodes, ..., and so on. The complexity of Stage 1 of the single row placement algorithm is

$$k_{FM} \cdot O(n_0) + 2k_{FM} \cdot O(n_0/2) + 4k_{FM} \cdot O(n_0/4) + \cdots = k_{FM} \cdot O(n_0 \log n_0)$$

If we set the balance deviation parameter s in the modified FM algorithm to be very small, the number of overlapping cells after Stage 1 is practically a small constant.[6] In this case, Stage 2 for overlap removal takes linear time in terms of n_0. Therefore, the complexity of each single row placement algorithm is $k_{FM} \cdot O(n_0 \log n_0)$. Note that $r \cdot n_0 = n$. Thus, the complexity of Phase 2 is

$$r \cdot k_{FM} \cdot O(n_0 \log n_0) = k_{FM} \cdot O(r \cdot n_0 \log n_0) = k_{FM} \cdot O(n \log n)$$

In order to obtain good results, the modified FM partitioning algorithm is executed $i1$ times for each partition in Phase 1 and $i2$ times for each partition in Phase 2. Therefore, the overall complexity of the algorithm is

$$i1 \cdot k_{FM} \cdot O(n \log n) + i2 \cdot k_{FM} \cdot O(n \log n) = (i1 + i2) \cdot k_{FM} \cdot O(n \log n)$$

Since $i1$, $i2$, and k_{FM} are all constants, the overall complexity of the algorithm is $O(n \log n)$ in practice.

4. Experimental Results

The algorithm proposed in this paper, AKROPOLIS, was implemented using C under Unix on Sun SPARC workstations. We tested the algorithm on the MCNC standard cell placement benchmarks. The description of these circuits is shown in Table 4.1. The first column represents the number of standard cells in the designs. The second and third columns represent the number of I/O pads and the number of the nets in the designs, respectively. Finally, the fourth column represents the number of rows used in our designs.

We compared our results with the ones produced by the TimberWolf6.0 placement and global routing package. In order to get a fair comparison of the placement solutions, we used the TimberWolf global router to route both our placement and the placement produced by TimberWolf.

---

[6] If we want to bound the number of overlapping cells by a constant c, we can set $s \leq 2^{\frac{c}{2 \cdot \log n_0}} - 1$.

circuit	cells	pads	nets	rows
fract	125	24	147	6
primary1	752	81	904	16
struct	1888	64	1920	20
industry1	2271	814	2593	24
primary2	2907	107	3029	28
biomed	6417	97	5742	44
industry2	12142	495	13419	69

Table 4.1: Description of the MCNC standard cell benchmarks

We followed the guideline given by MCNC about the design rules for these benchmarks. In particular, benchmarks *fract*, *struct*, *biomed* and *industry*2 were routed according to the MOSIS SCMOS 2.0μ design rules with $w1{=}s1{=}3\mu m$, $w2{=}3\mu m$, $s2{=}4\mu m$, where $w1$ and $w2$ represent the minimum width of horizontal and vertical wires, respectively, and $s1$ and $s2$ represent the minimum spacing between horizontal wires and between vertical wires, respectively. Benchmarks *primary*1 and *primary*2 were routed with $w1{=}w2{=}s1{=}s2{=}5\mu m$. Finally benchmark *industry*1 was routed with $w1{=}s1{=}s2{=}2\mu m$ and $w2{=}2.8\mu m$. All the net weights are set to one, since timing information was not available.

The comparison with TimberWolf6.0 is shown in Table 4.2. Columns 3 to 6 show the number of routing tracks, the total wirelength, the number of inserted feedthroughs, and the total area, respectively. The last column shows the runtime of each algorithm. Runtimes were recorded on a Sun SPARCII station with 32M memory. The *fast* parameter used by TimberWolf was set to zero for all circuits except *struct* and *industry*2 for which it was set to 5 (TimberWolf could not produce a solution after 48 hours, if the *fast* parameter was set to zero for these two examples).

From Table 4.2 it is clear that AKROPOLIS outperforms TimberWolf in every term compared. On average, AKROPOLIS reduces the total wirelength by 36.8%, reduces the chip area by 11.75%, reduces the number of inserted feedthroughs by a factor of 155 and reduces the runtime by a factor of 10.

The AKROPOLIS package can take four user specified parameters. Parameters $i1$ and $i2$ specify the number of times the modified FM partitioning algorithm is invoked for each bipartition in Phases 1 and 2, respectively. Parameters $s1$ and $s2$ specify the percentage of the area balance deviation allowed for each bipartitioning in Phases 1 and 2, respectively. Parameter $s1$ controls the rowlength difference. Parameter $s2$ controls the amount of overlap allowed in the first stage of Phase 2.

Table 4.3 shows the results produced by AKROPOLIS when the values of parameters $i1$ and $i2$ are adjusted. From Table 4.3 the trade-off between the quality of the placement solution and runtime becomes apparent. The larger the parameters $i1$ and $i2$ are, the better the quality of the results but the longer the running time. However, even for very small values of these parameters ($i1{=}i2{=}10$) the quality of the result is compared to TimberWolf while a factor of 147 speed-up is achieved. Running

circuit	algorithm	tracks	wirelength (μm)	feeds	area (mm^2)	time (s)
fract	TW	82	58710	187	0.575	185
	AKR	61	37580	36	0.483	89
primary1	TW	295	2018940	794	27.766	3374
	AKR	271	1421112	0	26.214[+]	1174
struct	TW	347	1017950	3243	8.617	102730
	AKR	253	468404	156	6.703	2725
industry1	TW	876	2704622	11178	20.708	57493
	AKR	782	1987649	5535	17.851	5611
primary2	TW	822	8305016	7609	109.321	50787
	AKR	757	5507915	78	101.457[++]	8936
biomed	TW	1148	4807366	15822	53.081	242284
	AKR	1075	3178506	1872	50.042	23889
industry2[+++]	TW	-	56228978	-	-	39922
	AKR	-	31565352	-	-	27137
ave. factor of improvement		1.18	1.61	154.6	1.14	10.03

[+] If the MOSIS SCMOS 2.0μ rules are applied, the *primary* 1 solution by AKROPOLIS occupies 20.65mm^2.
[++] If the MOSIS SCMOS 2.0μ rules are applied, the *primary* 2 solution by AKROPOLIS occupies 75.356mm^2.
[+++] Due to the memory limitation, *industry* 2 could not be routed by the TimberWolf global router.

Table 4.2: Comparison of the TimberWolf6.0 and AKROPOLIS placement results

AKROPOLIS with small $i1$ and $i2$ is very useful for fast and accurate area and delay estimation.

Table 4.4 shows the results produced by AKROPOLIS for different values of parameter $s1$. Smaller $s1$ leads to smaller rowlength difference, which in general results in smaller area and total wirelength.

5. Conclusions and Extensions

We have presented a new constructive approach to the placement problem for large row-based standard cell and gate array designs. The two-dimensional placement problem is transformed into a sequence of one-dimensional problems, and each of them can be solved independently using the generalized rectilinear distance facility location formulation. Extensive experimental results have confirmed that our algorithm is very efficient for large-scale IC designs and produces very high quality

i1	i2	wirelength (μm)	area (mm^2)	time (s)
10	10	563917	7.863	697
50	50	518122	7.490	1238
100	100	466981	6.878	1985
150	150	468404	6.703	2725

Table 4.3: AKROPOLIS placement results for the *struct* benchmark using different $i1$ and $i2$ parameter values ($s1$=0.5%, $s2$=25%).

s1	tracks	wirelength (μm)	rowlength dif. (μm)	area (mm^2)
0.5%	253	468404	71	6.703
1.0%	269	468264	214	7.118
1.5%	269	468504	262	7.251
2.0%	302	518105	338	7.804

Table 4.4: AKROPOLIS placement results for the *struct* benchmark using different $s1$ parameter values ($i1$=$i2$=150, $s2$=25%).

placement solutions. On average, AKROPOLIS reduces the total wirelength by 37%, reduces the chip area by 12%, reduces the number of inserted feedthroughs by a factor of 155, and reduces the runtime by a factor of 10 as compared to TimberWolf6.0.

Currently, we are enhancing the AKROPOLIS package to include several additional features. One extension is to handle the pre-placed macro-cells in the design. Special considerations are needed in the partitioning procedure used in Phase 1 and Phase 2 to handle the unequal row lengths due the existence of pre-placed macro-cells. Another extension is to handle the placement of I/O pads when their positions are not given. The I/O pad positions can be determined in Phase 1 where they are partitioned together with the cells at each iteration. An additional balance constraint is needed for the I/O pads during each partitioning so that the I/O pads are distributed evenly along the chip boundary. We also plan to dynamically update the weights of the nets, as the cells are partially placed, so that the interconnection delay can be further minimized.

Acknowledgments

The authors would like to thank Professor Carl Sechen and Bill Swartz for providing the TimberWolf package and their valuable help in our comparative study. This work is partially supported by National Science Foundation under grant MIP 9110511 and a research grant from UCLA Academic Senate.

References

[Ak81] Akers, S. B., "On the Use of the Linear Assignment Algorithm in Module Placement," *Proc. of the 18th ACM/IEEE Design Automation Conference*, pp. 137-144, 1981.

[Br77] Breuer, M. A., "A Class of Min-Cut Placement Algorithms," *Proc. 14th ACM/IEEE Design Automation Conf.*, pp. 284-290, 1977.

[CaFS70] Cabot, V., R. L. Francis, and M. A. Stary, "A Network Flow Solution to a Rectilinear Distance Facility Problem," *AIIE Trans.*, Vol. **2**, pp. 132-141, 1970.

[CoCa80] Cox, G. W. and B. D. Carroll, "The Standard Transistor Array (star), part II: Automatic Cell Placement Techniques," *Proc. of the 17th ACM/IEEE Design Automation Conference*, pp. 451-457, 1980.

[Do90] Donath, W. E., "Timing Driven Placement Using Complete Path Delays," *27t ACM/IEEE Design Automation Conference* , June 1990.

[DoJS92] Doll, K., F. M. Johannes, and G. Sigl, "Placement Improvement by Network Flow Methods," *International Workshop on Layout Synthesis*, Vol. **2**, pp. 179-182, May 1992.

[DuK85] Dunlop, A. E. and B. W. Kernighan, "A Procedure for Placement of Standard-Cell VLSI Circuits," *IEEE Trans. on Computer-Aided Design of Circuits and Systems*, Vol. **CAD-4**(1) pp. 92-98, Jan. 1985.

[FiMa82] Fiduccia, C. and R. Mattheyses, "A Linear Time Heuristic for Improving Network Partitions," *ACM/IEEE Design Automation Conf.*, pp. 175-181, 1982.

[FoFu62] Ford, L. R. and D. R. Fulkerson, *Flows in Networks*, Princeton Univ. Press, Princeton, N.J. (1962).

[GaVL91] Gao, T., P. Vaidya, and C. L. Liu, "A New Performance Driven Placement ALgorithm," *Proc. IEEE Int'l Conf. on Computer-Aided Design*, pp. 44-47, Nov. 1991.

[GaVL92] Gao, T., P. Vaidya, and C. L. Liu, "A Performance Driven Macro-Cell Placement Algorithm," *Proc. ACM/IEEE Design Automation Conf.*, June 1992.

[Gi62] Gilmore, P. C., "Optimal and Suboptimal Algorithms for the Quadratic Assignment Problem," *Journal of the Society for Industrial and Applied Mathematics*, Vol. **10**(2) pp. 305-313, June 1962.

[Go81] Goto, S., "An Efficient Algorithm for the two-Dimensional Placement Problem in Electrical Circuit Layout," *IEEE Trans. on Circuits and Systems*, Vol. **CAS-28**(1) pp. 12-18, Jan 1981.

[Ha70] Hall, K. M., "An r-dimensional Quadratic Placement Algorithm," *Management Science*, Vol. **17**, pp. 219-229, 1970.

[HaKu72] Hanan, M. and J. M. Kurtzberg, *Design Automation of Digital Systems, vol. 1: Theory and Techniques, M. A. Breuer ed., Chapter 5: Placement Techniques*, Prentice-Hall, Inc., Englewood Cliffs, N.J. (1972).

[HaNY87] Hauge, P. S., R. Nair, and E. J. Yoffa, "Circuit Placement for Predictable Performance," *International Conference on Computer-Aided Design*, pp. 88-91, Nov. 1987.

[HaWA76] Hanan, M., P. K. Wolff, and B. J. Agule, "Some Experimental Results on Placement Techniques," *Proc. of the 13th ACM/IEEE Design Automation Conference*, pp. 214-224, 1976.

[IoKB83] Iosupovici, A., C. King, and M. A. Breuer, "A Module Interchange Placement Machine," *Proc. of the 20th ACM/IEEE Design Automation Conference*, pp. 171-174, 1983.

[JaKu89] Jackson, M. and E. S. Kuh, "Performance-Driven Placement of Cell Based IC's," *26th ACM/IEEE Design Automation Conference*, pp. 370-375, June 1989.

[Ka83] Kang, S., "Linear Ordering and Application to Placement," *Proc. of the 20th ACM/IEEE Design Automation Conference*, pp. 457-464, 1983.

[KaCK82] Kambe, T., T. Chiba, S. Kimura, T. Inufushi, N. Okuda, and I. Nishioka, "A Placement Algorithm for Polycell LSI and its Evaluation," *Proc. of the 19th ACM/IEEE Design Automation Conference*, pp. 655-662, 1982.

[KhP77] Khokhani, K. H. and A. M. Patel, "The Chip Layout Problem: A Placement Procedure for LSI," *Proc. of the 14th ACM/IEEE Design Automation Conference*, pp. 291-297, 1977.

[KiGV83] Kirkpatrick, S., C. D. Gelat, and M. P. Vecchi, Jr., "Optimization by Simulated Annealing," *Science*, Vol. **220**, pp. 671-680, May, 1983.

[KlSJ91] Kleinhans, J. M., G. Sigl, F. M. Johannes, and K. J. Antreich, "GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization," *IEEE Trans. on Computer-Aided Design*, Vol. **10**(3) pp. 356-365, March 1991.

[KoTI83] Kozawa, T., H. Terai, T. Ishii, M. Hayase, C. Miura, Y. Ogawa, K. Kishida, N. Yamada, and Y. Ohno, "Automatic Placement Algorithms for High-Packing Density VLSI," *Proc. of the 20th ACM/IEEE Design Automation Conference*, pp. 175-181, 1983.

[Ku65] Kurtzberg, J. M., *Microelectronics in Large Systems*, Spartan Books, Washington, D.C. (1965).

[LaDi86] LaPotin, D. P. and S. W. Director, "MASON: A global floorplanning approach for VLSI Design," *IEEE Trans. on Computer-Aided Design*, Vol. **CAD-5**, pp. 828-837, 1986.

[Le90] Lengauer, T., *Combinatorial Algorithms for Integrated Circuit Layout*, John Wiley & Sons (1990).

[LiDu90] Lin, I. and D. H. C. Du, "Performance-Driven Constructive Placement," *27th ACM/IEEE Design Automation Conference*, pp. 103-106, June 1990.

[MaLi89] Marek-Sadowska, M. and S. P. Lin, "Timing Driven Placement," *International Conference on Computer-Aided Design*, pp. 94-97, Nov. 1989.

[OdHI87] Odawara, G., T. Hamuro, K. Iijima, T. Yoshino, and Y. Dai, "A Ruled-Based Placement System for Printed Wiring Boards," *Proc. of the 24th ACM/IEEE Design Automation Conference*, pp. 777-785, 1987.

[OdIW85] Odawara, G., K. Iijima, and K. Wakabayashi, "Knowledge-Based Placement Techniques for Printed Wiring Boards," *Proc. of the 22th ACM/IEEE Design Automation Conference*, pp. 616-622, 1985.

[PaSt82] Papadimitriou, C. H. and K. Steiglitz, *Combinatorial Optimization: Algorithm and Complexity*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1982).

[PiRa78] Picard, J. C. and H. D. Ratliff, "A Cut Approach to the Rectilinear Distance Facility Problem," *Operations Research*, Vol. **26**(3) pp. 422-433, 1978.

[PrLo88] Preas, B. and M. Lorenzetti, *Physical Design Automation of VLSI Systems*, The Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA (1988).

[QuBr79] Quinn, N. R. and M. A. Breuer, "A Force Directed Components Placement Procedure for Printed Circuit Boards," *IEEE Trans. on Circuits and Systems*, pp. 377-388, June 1979.

[Sc76] Schweikert, D. G., "A 2-Dimensional Placement Algorithm for the Layout of Electrical Circuits," *Proc. of the 13th ACM/IEEE Design Automation Conference*, pp. 408-416, 1976.

[ScUl72] Schuler, D. M. and E. G. Ulrich, "Clustering and Linear Placement," *IEEE Design Automation Workshop*, pp. 50-56, 1972.

[SeLe87] Sechen, C. and K. W. Lee, "An Improved Simulated Annealing Algorithm for Row-Based Placement," *Proc. IEEE Int'l Conf. on Computer-Aided Design*, pp. 478-481, 1987.

[SeS84] Sechen, C. and A. L. Sangiovanni-Vincentelli, "The TimberWolf Placement and Routing Package," *Proc. of the 1984 Custom Integrated Circuit Conference*, May 1984.

[SiDJ91] Sigl, G., K. Doll, and F. M. Johannes, "Analytical Placement: A Linear or a Quadratic Objective Function?," *Proc. of the 28th ACM/IEEE Design Automation Conference*, pp. 427-432, 1991.

[SrCK91] Srinivasan, A., K. Chaudhary, and E. S. Kuh, "RITUAL: Performance Driven Placement Algorithm for Small Cell ICs," *Proc. IEEE Int'l Conf. Computer-Aided Design*, pp. 48-51, 1991.

[StKK79] Stabler, E. P., V. M. Kureichik, and V. A. Kalashnikov, "Placement Algorithm by Partitioning fo Optimum Rectangular Placement," *Proc. of the 16th ACM/IEEE Design Automation Conference*, pp. 24-25, 1979.

[SuSh90] Sutanthavibul, S. and E. Shragowitz, "An Adaptive Timing-Driven Layout for High Speed VLSI," *27th ACM/IEEE Design Automation Conference*, June 1990.

[Yak92] Yang, Y. Y. and C. M. Kyung, "HALO: An Efficient Global Placement Strategy for Standard-Cells," *IEEE Trans. on Computer-Aided Design*, Vol. **11**(8)Aug. 1992.