GM_PLAN: A GATE MATRIX LAYOUT ALGORITHM BASED ON ARTIFICIAL INTELLIGENCE PLANNING TECHNIQUES

Yu Hen Hu

Department of Electrical and Computer Engineering
University of Wisconsin - Madison
Madison, WI 53706, U.S.A.

Sao-Jie Chen

Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan 10674, R.O.C.

ABSTRACT

In this paper, the gate matrix layout is formulated as a planning problem where a "plan" (the solution steps) is generated to achieve a "goal" (the gate matrix layout) that consists of subgoals interacting each other. Each subgoal corresponds to the placement of a gate to a slot, or to the routing of a net connecting gates. The interaction among subgoals is managed with two AI planning techniques: the hierarchical planning and the meta-planning. A new distance measure is defined to arrange the subgoals into prioritized classes in the hierarchical planning. Two meta-planning policies—graceful retreat and least impact—are used to decide which subgoal is to be achieved within the same priority class and how it can be achieved. In doing so, GM_Plan successfully combines the gate placement and net routing of the gate matrix layout into one process and has the potential to deliver better results.

INTRODUCTION

Gate matrix layout is a systematic CMOS layout methodology developed at Bell Laboratories [1]. In a gate matrix layout, the MOS transistors and i/o signal "gates" are realized in a vertical polysilicon wire and placed as column "slots" of a "gate matrix." An MOS transistor is formed when a horizontal diffusion layer wire overlaps the polysilicon (gate) column. And "nets" (wires) interconnecting "gates" are horizontally laid out in a metal layer and routed as row "tracks" of the gate matrix. Power and ground are also distributed using horizontal metal wires. As an example, the gate matrix layout and the symbolic representation of a CMOS NAND circuit are shown in Fig. 1(a) and 1(b). Its implementation requires two major tasks: placing gates in a linear array of slots and routing nets on horizontal tracks. Since the column (slot) number is fixed, the design objective is to select an appropriate ordering of gates so that the total number of tracks taken by the nets is kept to a minimum. It has been shown that this is a difficult (NP-Complete) problem [2] when the number of gates to be placed becomes large. Hence, heuristics are used to derive practical and efficient gate matrix layout algorithms.

Two major approaches have been presented in the past. The first approach aims at finding an optimal "gate assignment" so that the total number of tracks demanded is minimized. After the gate ordering is determined, a left-edge algorithm [3] is applied to perform the task of track assignment (net routing) [4-9]. The second approach attacked the gate matrix layout problem by first "assigning nets" to tracks; the gate ordering was subsequently decided [10,11].

These existing gate matrix layout algorithms share some common weaknesses. Hence, some improvements are proposed:

1. Although the gate matrix layout style has the potential to "combine the placement and routing into one process" [12, the existing methods often ignore this fact and regard them as two separate subproblems solved one after another. In the presented GM_Plan, the processes of gate placement and net routing are combined into the same problem solving loop.

2. Premature commitments of gate placement or net routing may result in a suboptimal solution. In the above methods, the tasks of placing a gateor routing nets to racks are accomplished using a *greedy* algorithm that is relatively insensitive to global constraints. In GM_Plan, *look-ahead* planning strategies are incorporated to help avoiding suboptimal solutions.

This research is supported in part by National Science Foundation under contract MIP-8896111, and in part by DARPA under contract MDA 903-86-C-0182.

Problem Statement

A CMOS circuit can be described by a set of input/output nodes and transistor "gates" $\{g(i)\}$, and a set of nets $\{n(i)\}$ connecting nodes and gates. The set of nets connecting to each gate (or node) forms a gate-net set. Correspondingly, all associated gates of a net form a net-gate set. A table listing each gate and the corresponding gate-net set is called a GN (Gate-Net) table. By the same token, a table listing each net and the associated net-gate set is called an NG (Net-Gate) table. A "net-group" $N(g)$ is the set of nets connecting to a gate g. Similarly, a "gate-group" $G(n)$ is the set of gates connected to a net n.

We shall assume the NG (or GN) table is given at the beginning. Depending on different conventions, the NG and GN tables can be represented in either a *fixed-net* [6] or a *dynamic-net* representation [9]. We prefer to use a dynamic-net like representation. For example, given the two-input NAND circuit as shown in Fig. 1, we have: NG_n: (n1; a, b, c), and GN_n: (a; n1), (b; n1), (c; n1); $N(b) = \{n1\}$ and $G(n1) = \{a, b, c\}$.

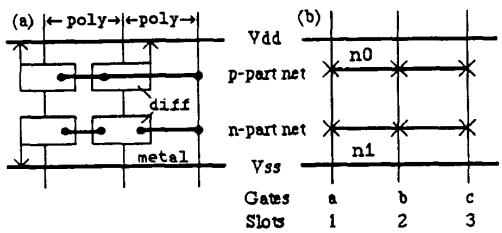

Fig. 1. (a) Symbolic Layout of a two-input NAND Circuit
(b) Symbolic Representation of the NAND Circuit.

With this abstract representation, the gate matrix layout problem can be formally defined as follows:

Given an NG table, a set of linearly aligned array of slots $\{s(i)\}$, and a set of empty tracks $\{t(i)\}$, find a one-to-one mapping from the set of gates $\{g(i)\}$ to the set of slots $\{s(i)\}$:

$$f1:\ g(i) \longrightarrow s(i)$$

and a second mapping from the set of nets $\{n(i)\}$ onto the set of tracks $\{t(j)\}$:

$$f2:\ n(i) \longrightarrow t(j)$$

so that the total number of tracks required is minimized.

GATE MATRIX LAYOUT: A PLANNING APPROACH

In the presented GM_Plan, the gate matrix layout problem is formulated as a planning problem consisting of two interwoven subgoals: gate placement and net routing. Two Artificial Intelligence planning strategies are used in GM_Plan: the hierarchical planning and the meta-planning policies.

ISCAS '89

Hierarchical Planning

Planning on different levels of abstraction, or "hierarchical planning" [13,14], is a "divide-and-conquer" technique based on the partitioning of a problem into subproblems of different priority classes (abstraction levels). In GM_Plan, a novel "distance measure" and a notion of "nearest-neighbor group" is proposed to facilitate the hierarchical planning.

A new distance measure: For serially connected transistors, permutation of their order does not affect the correctness of the specification. Similar argument can also be applied to parallel connection transistors (There are special cases where this permutation is prohibited: *e.g.*, interchanging two transistors may cause charge sharing problem, or the driving capability of the transistors will be affected due to this permutation of orders). Hence, it would be meaningless to measure the "distance" between two gates as the difference between their respective column (slot) numbers in the gate matrix. Since the problem requirement is to connect all gates within a net-gate set, it seems more appropriate to consider two gates being *adjacent* if they are linked *directly* by at least one net. Therefore, we propose a new definition of *distance* for the categorization of gates into different priority classes:

Definition: *Distance between two gates*. Let $d(g1,g2)$ denote the *distance* between two different gates $g1$ and $g2$; then

a) $d(g1,g2) = 1$ if $\exists$ a net connecting both $g1$ and $g2$, and

b) $d(g1,g2) = n\ (n \geq 2)$ if

(1) $d(g1, g2) \neq 1$ to $n-1$ and

(2) $\exists$ another gate, say, gi so that

$d(g1,gi) = n-1$, and $d(g2,gi) = 1$; or

$d(g1,gi) = 1$, and $d(g2,gi) = n-1$.

This definition can be generalized to define the distance between two sets of gates:

Definition: *Distance between two gate-sets*. Let $G(1)$ and $G(2)$ be two non-empty sets of gates with empty intersection, and $d(G(1),G(2))$ denote the *distance* between $G(1)$ and $G(2)$; then $d(G(1),G(2)) = n$ if $\exists\, g1 \in G(1)$ and $g2 \in G(2)$, $Min.[d(g1, g2)] = n$.

For illustration purpose, in Fig. 2, the distance between gate $g1$ and $g2$ is one. *i.e.*, $d(g1,g2) = 1$. Also, the distance between the gate set $G(1)$ and $G(2)$ is one, *i.e.*, $d(G(1),G(2)) = 1$. Similarly, $d(g1,g3) = 2$, and $d(G(1), G(3) = 2$.

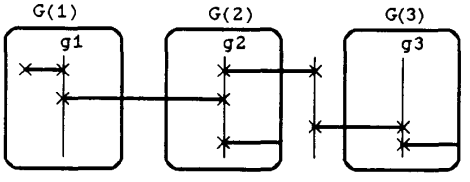

Fig. 2. Gate Sets of Distance One and Two.

Nearest-Neighbor Group: Using the new distance measure, the set of unplaced gates can be classified into different priority classes, each of which contains gates having the same distance from the PLaced Gate Set *(PLGS)*. The levels within this hierarchy determine (roughly) the ordering of gates. All the distance-one gates of the *PLGS* form a *Nearest-Neighbor Group (NNG)*, *i.e.*, $NNG = \{g^* \mid g^* \notin PLGS$, and $d(g^*, PLGS) = 1\}$. The *NNG* contains gates that are most relevant to the *PLGS* and hence is the most critical subgoal at the current time.

The contents in the *NNG*, and the remaining levels in the hierarchy are incrementally updated as more and more gates are placed. Whenever a gate is placed, it "pulls" its distance-1 neighbors from a lower priority class into the current *NNG*. This change, in turn, ripples to the rest of levels in the hierarchy by updating their respective contents. To minimize the computation overhead, in the real implementation, only the *NNG* gates with respect to the placed gates are identified. Hence, the updating of other priority classes can be accomplished incrementally by using *NNG*.

Meta-Planning Control Policies

Meta-planning is a technique concerned with the control of planning decisions—knowing when and when not to make commitments in a subplan to achieve a subgoal [15]. To avoid premature commitments, a control policy ("least-commitment") was used [15] which will not make a decision (commit certain resources) until compelling evidence appears. However, when applied to the gate matrix layout problem, this policy leads to too many deferred commitments and renders the later decision-making process more difficult if not infeasible. This is reminiscent of the two meta-planning control policies for switchbox routing problems [16], which is applicable to the current gate matrix layout problem.

The first one, *Graceful Retreat (GR)*, is the policy of selecting a subgoal that is the most critical, *i.e.*, one that has the smallest solution space, and achieving it before other subgoals. The idea is that since all subgoals must be accomplished to achieve the overall goal, the most critical one must be addressed first, even at the expense of consuming some resources of potential use by other subgoals.

The second one, *Least Impact (LI)*, is the policy of choosing a solution among many feasible ones to the subgoal so as to have the least impact on the remaining subgoals. The rationale is to preserve as much flexibility or as many resources as possible for achieving future subgoals.

These *GR* and *LI* policies are incorporated in our "Gate Matrix layout Planning model" (GM_Plan), illustrated in Fig. 3.

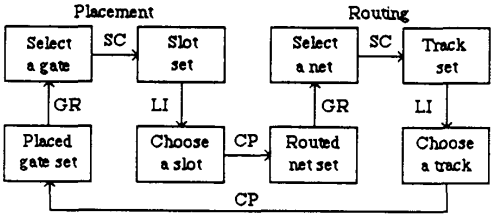

Fig. 3. Planning Model for Gate Matrix Layout;
SC: Solution Construction GR: Graceful Retreat
CP: Constraints Propagation LI: Least Impact.

In this model, two subproblems, gate placement and net routing, are iteratively solved for each gate and its corresponding nets. At the beginning of each iteration, a gate is selected from a set of unplaced gates and placed under the control of the *GR* and *LI* policies. After a gate is placed, constraints are propagated to the environment for net routing. These two meta-planning policies are used again to choose the most critical net that will be routed on the least impact track, and the environment is updated by constraint propagation. Then, the tasks of gate placement and net routing proceed alternatively. Heuristics and constraints are used to guide the search, to prune the search space, and to handle the highly interactive environment of placement and routing in the gate matrix layout.

The Algorithm—GM_Plan

The steps taken by GM_Plan are outlined below; the algorithm is summarized in Fig. 4.

Step 0. *Prepare NG, GN tables.*

Step 1. *Unique Placement and Routing*: Select a seed gate from the given GN table, and assign it to a slot. Also, route each net of the seed gate on a track.

Step 2. *Form the nearest-neighbor group:* Select the *NNG* of the seed gate from the set of unplaced gates (*NNG* are gates that have at least one direct connection to the gate(s) already placed).

Step 3. *Repeat the steps (3a) to (3c) until all gates are placed.*

(3a) *Gate Placement:* This step consists of two procedures.

1. Select-gate Procedure is to decide which among all the yet-to-be-placed gates should be selected and placed next. The *GR* policy is used to select a gate that is the "most critical" subgoal to be achieved at the current stage.

a) Select the gate g so that the intersection of $N(g)$ and $N(PLGS)$ has the maximum number of nets.

b) If there is more than one such gate, select the gate that is of distance one to the seed gate.
c) If there is more than one such gate or no such gate, select the gate that has more nets per gate (This is also the rule used for selecting the seed gate at the beginning).
d) If there is more than one such gate, arbitrarily select one.

2. Assign-slot Procedure: Once a gate is selected, it will be placed into a slot. In GM_Plan, the *LI* policy governs the slot assignment process with constraint pruning and heuristic search. Constraints are used to prune the search space by eliminating unlikely slots. The heuristic search is accomplished by evaluating a cost function *f* for each of the remaining slots. This cost function consists of a linear combination of four factors:

$$f = g1 + h1 + h2 + h3$$

where *g1* is called the fixed-net connection cost, *h1* the fixed-net expansion cost, *h2* the floating-net expansion cost, and *h3* the floating-net connection cost. Here, *fixed nets* are nets that have already been routed on a track, and *floating nets* are those that have not been routed. *Connecting costs* are the (new) wire lengths needed to route nets, and *expansion costs* are the possible increase of (old) wire lengths caused by the current slot selection.

a) Choose a slot that when the selected gate is placed, no additional tracks will be needed immediately.
b) If more than one slot satisfies (a), go to (c); otherwise, if no such slot exists (*i.e.*, some tracks have to be added no matter what slot is assigned), choose the slot that minimizes the number of tracks to be added.
c) If more than one slot satisfies (a) or (b), choose the slot that
(1) facilitates a maximum number of immediate connection of nets of the selected gate but with less *g1*;
(2) disturbs less the existing net route (with less *h1*);
(3) will facilitate the connection of floating nets (with less *h2*);
(4) will disturb less the floating net route (with less *h3*).
Each of above criteria is a term of the cost function *f* used in GM_Plan. Thus, step (c) is to choose a slot that minimizes *f*.
d) If more than one slot satisfies (c), choose the slot that is the closest to one of the two ends of the gate matrix layout.
e) If there is more than one such slot, arbitrarily choose one.

(3b) *Net Routing:* The fixed nets are routed using constraints first. The "select-net" and the "assign-track" procedures are applied subsequently to route part of the floating nets.

1. Select-net Procedure (1): The *GR* policy is applied to select the most critical floating net to be routed. The criticality is measured in terms of the difficulty to route a particular net.

a) Select the net that has the largest number of gates per net.
b) If more than one net satisfies (a), select the net that has the largest number of gates in the intersection between its net-gate set and the placed gate-set.
c) If there is more than one such net, arbitrarily select one.

2. Select-net Procedure (2): Since not all floating nets need to be and can be routed immediately, floating nets with one or more of the following properties shall not be selected. (Instead, their routing is postponed until the "reroute" or the final "wrap-up-route" phase, where the "lose-end" nets are routed.)

d) Discard nets that have just one gate placed, or
e) Discard nets that have a larger "equivalent set of possible tracks" to route.

3. Assign-track Procedure: The *LI* policy is applied to choose the best solution from available tracks.

a) Choose the track that is the most compact, *i.e.*, the one that wastes the least resource of area, to route.
b) If more than one such track exists, choose the track that fits fewer nets.
c) If more than one such track exists, arbitrarily choose one.

(3c) *Reroute*:: The *LI* policy used in the "assign-track" is the practice of a greedy principle: to accomplish as much as possible the task at hand. But greedy heuristics may lead to local minima and suboptimal solutions. On the other hand, effective greedy heuristics can significantly reduce the cost of combinatorial searches. In the implementation, trade-off between too much greediness (to route all the floating nets) and total inefficiency (no floating nets are routed) has to be made. We compromise this trade-off by using a "reroute" procedure (which is equal to "select-net procedures (1) and (2)" plus "assign-track") after all the distance-one gates of the first seed gate are placed

Step 4. *Wrap-up routing.* Perform "select-net procedure (1)" and "assign-track" after all the gates are placed.

```
0   Prepare_Table (NG, GN);                    /* Step 0 */
1   Unique_pl&route ()                         /* Step 1 */
2     {   select-seed (sg);
3         unique-place (sg);
4         unique-route (N(sg));
5         PLGS = {sg};
6     }
7   Form_NNG ()                                /* Step 2 */
8     {   NNG = G(N(sg)) \ {sg};  }
9   CRGS = NNG;
10  Layout ()
11    { While (CRGS != NULL) repeat
12      {  Place_gate ()                       /* Step 3a */
13           {   select-gate (g, CRGS) with GR policy;
14               assign-slot (g, slot) with LI policy;
15           }
16         Route_net (N(g))                    /* Step 3b */
17           {   select-net (n, N(g)) with GR policy;
18               assign-track (n, track) with LI policy;
19           }
20         if (end of NNG) reroute (N(sg));    /* Step 3c */
21         PLGS = PLGS ∪ {g};
22         CRGS = {CRGS ∪ G(N(g)) } \ PLGS;
23      }
24    }
25  Wrap_up_route (all-nets).                  /* Step 4 */
```

Fig. 4. The GM_Plan Algorithm.

COMPLEXITY ANALYSIS AND PERFORMANCE EVALUATION

An optimal gate matrix layout, in general, can be obtained only in factorial time, and hence, is an NP-complete problem [2]. Thus, all existing gate matrix algorithms, including GM_Plan, are based on heuristics. Two criteria are used in this paper to compare GM_Plan with others. First, a theoretical complexity analysis is conducted. Next, the performance of GM_Plan is compared with known best results.

Complexity Analysis

Assume that n = # of gates, m = # of nets, α = max # of gates per net, β = max # of nets per gate, and tn = # of tracks used for the gate matrix layout, clearly $m \geq tn \geq \beta$. Then, refer to the GM_Plan algorithm (see Fig. 4), we have:

Step 1: The procedure "select-seed" (line 2),which selects a seed-gate from the GN table, needs O(n) time. The "unique-place" and "unique-route" (lines 3-4) both need constant time.

Step 2: The *NNG* (line 8) can be formed in $O(\alpha*\beta)$ time.

Step 3a: The "select-gate" (line 13) requires O(n*m) time units. This is because the intersection of *N(g)* and *N(PLGS)* is of O(m) time, and there are at most n gates to be evaluated. The "assign-slot" (line 14) needs O(n*tn) time, for each gate may connect to O(tn) nets, and there are at most O(n) slots to assign.

Step 3b: The "select-net" (line 17) needs at most $O(\beta)$ time. Also, the "assign-track" (line 18) searches tn tracks and hence needs O(tn) time to route a current net. But it has a hidden cost of O(n*tn) in order to check the routability of each track.

Step 3c and 4: The "reroute" (line 25) needs O(n*tn) time, and the same for the "wrap_up_route" in line 20.

(217) 333-4789

From the above analyses, the maximum time required is O(n*m) and the iteration count is O(n). Thus, the overall time complexity of GM_Plan is $O(n^2*m)$.

As for the space complexity, we use two double-linked lists (taking O(n) memory space) to store the placed gates and available slots respectively, and a single-linked list (taking O(tn) space) to store the routing track information. In total, the memory space taken is O(n*tn).

Performance Evaluation

The GM_Plan has been implemented in C language on a VAX 11/750 running UNIX (version 4.3 BSD). The code length is around 3,000 lines. It accepts an NG table as input and generates a character-based symbolic gate matrix layout.

Sixteen benchmark circuits were tested and the results are listed in Table 1. The first four examples are from Heinbuch's book [17]: "v4000" is a 4x1 mux, "v4050" a 2x4 decoder, "v4090" a 3x8 decoder, and "v4470" a 4-bit comparator. There are another four examples from Wing and Huang [20]: "w2" is ITT1, "w3" is a 4-bit ALU, "w4" is ITT2, and "wan" a full-adder. The remaining examples are collected from published literature and their sources are given in Table 1.

Table 1.—Gate Matrix Layout Results with GM Plan.

Name	References	# of Nets (m)	# of Gates (n)	Lower Bound (β)	Known Solution	Planning Solution (tn)	CPU time VAX 750 (sec)
v4000	[17]	10	17	5		7	2.2
v4050	[17]	13	16	5		6	1.6
v4090	[17]	23	27	9		11	4.8
v4470	[17]	37	47	5		14	9.3
vc1	[18]	15	25	9	9	10	3.9
vl	[19]	6	8	3	3	3*	0.5
vw1	[5]	5	7	4	4	4*	0.5
vw2	[6]	8	8	3	5	5*	0.7
w1	[5]	18	21	4	4	4*	1.9
w2	[20]	48	33	14	14	14*	8.8
w3	[20]	84	70	11	23	21	25.8
w4	[20]	202	141	18	36	46	105.7
wan	[20]	8	7	6	6	6*	0.6
wli	[7]	11	10	4	4	6	0.6
wsn	[21]	17	25	4	8	8*	3.2
x0	[22]	40	48	6	15	15	11.3

(* means optimal solution.)

Of the last 12 examples in Table 1, optimal solutions are reached in 7 of them. The GM_Plan solution of "w3" is 21-track, better than the known 23-track solution obtained from Wing [20]. The solutions of the other six examples ("vl," "vw1," "vw2," "w1," "wan," and "wsn") are equal to the known ones. Only three examples ("w4," "wli," and "vc1") generated higher-track outputs.

CONCLUSION

In this paper, we have presented an $O(n^2*m)$ polynomial time *GM_Plan* algorithm for the gate matrix layout. This method is based on an AI planning paradigm. Using the hierarchical planning technique, and two meta-planning policies, this new method is more sensitive to the interactions among subgoals—those of placing gates and routing nets. Results from running benchmark examples have been very encouraging. Further research is underway aiming at following directions:

1. The implementation of an efficient backtracking facility to further reduce the chances of making premature commitment during the problem solving process.
2. The use of iterative learning techniques to further improve the quality of the output [23].

ACKNOWLEDGMENTS

The authors wish to thank Dr. O. Wing and Mr. S. Huang of Columbia University for providing examples, and many helpful comments. Our thanks also extend to Dr. C. K. Cheng of U.C. San Diego, Dr. J. F. Beetem and Dr. B. N. Mayo of U.W. Madison, and Dr. J. T. Li of Advanced Micro Devices for many useful comments on this manuscript.

REFERENCES:

[1] A. D. Lopez and H-F. S. Law, "A Dense Gate Matrix Layout Method for MOS VLSI," *IEEE Tr. Electron Devices*, vol. ED-27, no. 8, pp. 1671-1675, Aug. 1980.

[2] T. Kashiwabara and T. Fujisawa, "An NP-Complete Problem on Interval Graph," *Proc. ISCAS*, pp. 82-83, 1979.

[3] A. Hashimoto and J. Stevens, "Wire Routing by Optimizing Channel Assignment Within Large Apertures," *Proc. 8th DA Workshop*, pp. 155-169, 1971.

[4] T. Ohtsuki, H. Mori, E. S. Kuh, and T. Fujisawa, "One-Dimensional Logic Gate Assignment and Interval Graph," *IEEE Tr. Circuits and Systems*, vol. CAS-26, no. 9, pp. 675-684, Sep. 1979.

[5] O. Wing, "Automated Gate Matrix Layout," *Proc. ISCAS*, pp. 681-685, 1982.

[6] O. Wing, S. Huang and R. Wang, "Gate Matrix Layout," *IEEE Tr. CAD*, vol. CAD-4, no. 3, pp. 220-231, July 1985.

[7] J. T. Li, "Algorithms for Gate Matrix Layout," *Proc. ISCAS*, pp. 1013-1016, 1983.

[8] C. K. Cheng, "Decomposition Algorithm for Linear Placement and Application to VLSI Design," *Proc. ISCAS*, pp. 1047-1050, 1985.

[9] D. K. Hwang, W. K. Fuchs, and S. M. Kang, "An Efficient Approach to Gate Matrix Layout," *IEEE Tr. CAD*, vol. CAD-6, no. 5, pp. 802-809, Sep. 1987.

[10] T. Asano and K. Tanaka, "A Gate Placement Algorithm for One-Dimensional Arrays," *J. of Information Processing*, vol. 1, no. 1, pp. 47-52, 1978.

[11] S. Huang and O. Wing, "Improved Gate Matrix Layout," *Proc. ICCAD*, pp. 320-323, 1986.

[12] T. C. Hu and E. S. Kuh, "Theory and Concepts of Circuit Layout," in *VLSI Circuit Layout: Theory and Design*, eds. T. C. Hu and E. S. Kuh, IEEE Press, New York, 1985.

[13] E. D. Sacerdoti, "Planning in a Hierarchy of Abstraction Spaces," *Artificial Intelligence*, 5, pp. 115-135, 1974.

[14] M. Stefik, "Planning with Constraints, (MOLGEN: Part I)," *Artificial Intelligence*, 16, pp. 111-139, 1981.

[15] M. Stefik, "Planning with Meta-Planning, (MOLGEN: Part II)," *Artificial Intelligence*, 16, pp. 141-169, 1981.

[16] W. P. C. Ho, D. Y. Y. Yun, and Y. H. Hu, "Planning Strategies for Switchbox Routing," *Proc. ICCD*, pp. 463-467, 1985.

[17] D. V. Heinbuch, *CMOS3 Cell Library*, Addison-Wesley, Reading, Massachusetts, 1988.

[18] Y. C. Chang, S. C. Chang, and L. H. Hsu, "Automated Layout Generation Using Gate Matrix Approach," *Proc. 24th DA Conf.*, pp. 552-558, 1987.

[19] H. W. Leong, "A New Algorithm for Gate Matrix Layout," *Proc. ICCAD*, pp. 316-319, 1986.

[20] O. Wing and S. Huang, Private correspondence, Jan. to Aug. 1988.

[21] O. Wing, "Interval-Graph-Based Circuit Layout," *Proc. ICCAD*, pp. 84-85, 1983.

[22] K. Nakatani, T. Fujii, T. Kikuno, and N. Yoshida, "A Heuristic Algorithm for Gate Matrix Layout," *Proc. ICCAD*, pp. 324-327, 1986.

[23] S. J. Chen and Y. H. Hu, "GM_Learn: An Iterative Learning Algorithm for CMOS Gate Matrix Layout," *Proc. ISCAS*, 1989.