VLSI Design
1994, Vol. 2, No. 3, pp. 241–257
Reprints available directly from the publisher
Photocopying permitted by license only

SEGMA: A Simulated Evolution Gate-Matrix Layout Algorithm

CHI-YU MAO and YU HEN HU
Department of Electrical and Computer Engineering, University of Wisconsin-Madison, Madison, WI 53706 U.S.A.

(Received December 3, 1992, Revised March 17, 1993)

In this paper, we present a Simulated Evolution Gate-Matrix layout Algorithm (SEGMA) for synthesizing CMOS random logic modules. The gate-matrix layout problem is solved as a one-dimensional transistor gates placement problem. Given a placement of all the transistor gates, simulated evolution offers a systematic method to improve the quality of the layout that is measured by the number of tracks needed for the given netlist. This is accomplished by identifying a subset of gates whose relative placements are deemed "poor quality" according to a heuristic criterion. By rearranging the placement of these identified subsets of gates, it is hoped that a gate placement with better quality, meaning fewer tracks, may emerge. Since this method enables the current "generation" of gate placement to evolve into a more advanced one in a way similar to the biological evolution process, this method is called *simulated evolution*. To apply simulated evolution to solve the gate-matrix layout problem, we propose a novel heuristic criterion, called randomized quality factor, which facilitates the judicious selection of the subset of poor quality gates. Several carefully devised and tested strategies are also implemented. Extensive simulation results indicate that SEGMA is producing very compact gate-matrix layouts.

Key Words: *Gate-matrix layout, Simulated evolution, Stochastic search method, Random logic, Module generation*

1 INTRODUCTION

Gate-matrix layout [1] is a systematic CMOS layout methodology. With its regular structure and relatively high gate density, it has been adopted by a number of automatic layout and leaf-cell module generation systems [2]. In this paper, we propose a novel approach to solve the gate-matrix layout problem based on a recently introduced combinatorial optimization paradigm called simulated evolution (SE). With simulated evolution, a design (gate-matrix layout) is iteratively refined in a way similar to the biological evolution process: Each generation (design version) evolves to the next by a) dismantling a portion of the current solution; b) reconstructing a new design; and c) evolving to the new design if it satisfies certain acceptance criteria. Otherwise the new design is discarded and a new trial will be made. In each iteration of a SE based algorithm, two key steps are performed:

(a) Selection: A portion of the current design will be identified as "bad genes" according to certain heuristic criteria. This portion of design is blamed for contributing to the poor quality of the overall design, and hence it will be dismantled.

(b) Evolution: A new design will be constructed based on the remaining partial design. Evolution occurs if the new design satisfies the acceptance criteria.

Depending on different applications, the actual implementations of the SE method may be quite different. The SE method has been applied to the standard cell placement problem in a package called ESP [3]. In ESP, the "goodness" of each cell is defined as the average of the net placement value of all the nets connecting to that cell. The selection of a cell to be re-placed is accomplished by comparing the goodness of that cell with a random number. Hence the number of cells being selected in each iteration is random. The evolution (allocation) phase is further divided into early, intermediate, and final stages with different heuristic strategies. Later, in [4], a divide-and-conquer strategy is employed to reduce the search complexity. In applying the SE method to partitioning problems, Saab and Rao [5] suggested

different implementation strategies to handle a number of constraints. A new design is generated by two subsequent steps: migration and distribution. A new stopping heuristic is also introduced.

Lin *et al.* proposed a simulated evolution router SILK to solve switchbox routing problems [6]. In SILK, the cost of each net also includes the number of violations of the routing rules. A feasible solution is generated only when there are no rule violations. A post-optimization is included to reduce the total wire length and the number of vias after a feasible solution is reached.

The simulated evolution algorithm is closely related to Genetic algorithm (GA) [7]. Fourman [8] and Cohoon and Paris [9] have applied GA to solve various VLSI system design problems. Another relevant iterative improvement approach is Simulated Annealing (SA) [10]. The relations among SE, SA, and GA have been discussed in [3].

SEGMA is an application of SE to permutation problems. Two novel search strategies have been developed: First, in SEGMA, a new selection criterion called randomized quality factor is developed. The quality factor is a heuristic criterion for evaluating the quality of placement of individual gates. By randomizing the quality factor, we are able to blend the gradient based greedy search strategy with the random search strategy inherent in the SE algorithm. The purpose is to preserve search efficiency while enhancing the chance to reach a globally optimal solution. Second, in SEGMA, evolution to a new generation will take place as long as the quality of the new design (generation) is equal to or better than the current one. We note in previous SE implementation only better new designs were accepted. Intuitively, accepting an equal quality design allows SEGMA to explore more design space, and hence enhance the opportunity to converge to a globally optimal solution. Again, we are seeking a compromise between search efficiency and globally optimal solution. Extensive simulation results indicate these novel strategies appear to be very effective for enhancing the performance of SEGMA at moderate computing overhead.

The rest of this paper is organized as follows. The gate-matrix layout problems are characterized in section II. Several existing gate-matrix layout methods are reviewed in this section. In section III, SEGMA is presented to solve CMOS gate-matrix layout problems. Convergence analysis and benchmark experiments are discussed in section IV. Empirical evaluations of different implementation strategies of SEGMA are presented in section V.

2 THE GATE MATRIX LAYOUT PROBLEM

The symbolic representation of the gate-matrix layout of a two-input NAND circuit in Figure 1(a) is depicted in Figure 1(b). The transistors are separated into two groups. P-channel MOS transistors are situated at the top portion of the layout. N-channel MOS transistors are at the bottom portion. The orientations of all transistors are arranged such that the diffusion and metal wires are laid horizontally and polysilicon wires vertically. Three segments of metal wire *m1, m2* and *m3* enable the connection specified in Figure 1(a). The metal-diffusion contacts and metal-polysilicon contacts are represented by the same symbol of small black squares.

In the gate-matrix layout style, the layout can be regarded as a transistor matrix using polysilicon and metal wires to connect corresponding transistors. In the terminology convention of MOS circuits, the vertical polysilicon wire corresponds to the *gate* of an MOS transistor or to an output. There are three "gates" *a*, *b* and *y* in Figure 1(b). The diffusion areas on both sides of each gate form the drain and source of the MOS transistor respectively.

An abstract representation of the layout in Figure 1(b) is shown in Figure 1(c). The vertical lines stand for gates and horizontal lines for metals. No two different metal wires can be connected or overlap with each other. The circles in the intersection are metal to diffusion, or metal to polysilicon contacts. The validity of this simplified representation is explained in [11]. By convention, the horizontal wires are called nets in this abstract gate-matrix layout representation. The vertical wires are called gates.

D.K. Hwang *et al.* proposed the other representation of nets for gate-matrix layout, called *dynamic netlist* [12]. In order to demonstrate how to apply SE to the gate-matrix layout problem and to compare with our previous gate-matrix layout algorithms [19, 30], we use static netlist representation in this paper.

The realization of a gate-matrix layout requires the solution of two sub-problems: placing gates in a linear array of slots and routing nets along horizontal tracks. Since the number of gates (slots) is fixed, the design objective is to select an appropriate ordering of gates to minimize the total number of tracks needed to route all the nets.

2.1 Definitions

To facilitate further discussion, a number of terms related to gate-matrix layout will be defined below.

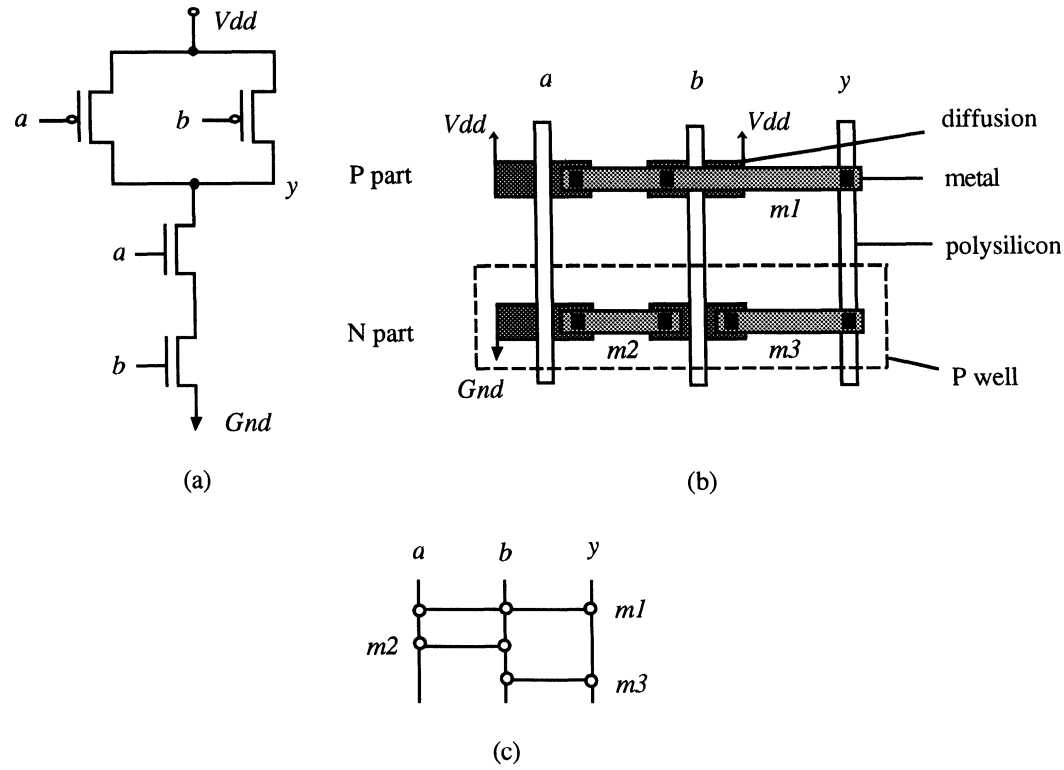


FIGURE 1 Representations of gate-matrix layout. (a) CMOS NAND logic circuit. (b) Symbolic representation of gate-matrix layout. (c) Simplified representation of gate-matrix layout.

Definition 1: $G = \{g_1, g_2, \ldots, g_n\}$ is the set of gates (transistors of MOS FET) in a gate-matrix layout. The number of gates is denoted by n.

Definition 2: $N = \{n_1, n_2, \ldots, n_m\}$ is the set of nets in a gate-matrix layout. The number of nets is denoted by m.

Definition 3: $G(n_i)$ is the set of gates to which the net n_i will connect. $N(g_i)$ is the set of nets that connect to the gate g_i. The minimum length of n_i equals to $|G(n_i)| - 1$ where $|G(n_i)|$ is the number of elements in the set $G(n_i)$.

Definition 4: S is the set of slots to which the gates will be assigned. In particular, $|S| = n$. $s_i = 0, 1, \ldots, n - 1$.

Definition 5: f: $G \rightarrow S$ is a one to one mapping function that assigns a slot for each gate. In particular, $f(g_i) = s_i$.

Definition 6: $L(n_i; f) = f(g_R) - f(g_L)$ is the length of the net n_i under the mapping function f, where g_R and g_L are rightmost gate (with the largest slot number) and the leftmost gate (with the smallest slot number) respectively.

Definition 7: $R(g_i; f)$ is the set of nets that physically cross over the gate g_i without connecting to g_i according to the mapping function f.

Definition 8: $H(g_i; f) = |N(g_i)| + |R(g_i; f)|$ is defined as the height of gate g_i. $H(f) = \max H(g_i; f)$ is the overall track usage of the corresponding gate-matrix layout according to the mapping function f.

With the above definitions, the gate-matrix layout problem can be formally stated as follows: Given a netlist specified by a NG set, find a mapping function f such that $H(f)$ is minimized.

Lower bound of track number: Note that

$$H(f) = \max_{g_i \in G}\{|R(g_i; f)| + |N(g_i)|\} \geq \max_{g_i \in G}\{|N(g_i)|\} \quad (2.1)$$

Hence $\max\{|N(g_i)|\}$ serves as a lower bound of the gate-matrix layout. If a gate assignment function f satisfies $H(f) = \max_{g_i \in G}\{|N(g_i)|\}$, then it is an optimal gate placement.

2.2 Existing Gate-Matrix Layout Methods

2.2.1 Constructive methods Constructive methods produce a complete layout based on a partial layout. Constructive algorithms are typically divided into the following classes: (1) graph-theoretic approach, (2) hierarchical approach, (3) branch-and-bound technique, and (4) artificial intelligence (A.I.) approach.

(1) Graph-theoretic approach: Ohtsuki *et al.* [13] used a graph-theoretic approach and formulated the problem into an interval graph problem. Wing [14, 11] proposed a solution by finding a minimal augmentation to transform a connection graph, which is derived from the topology of the given circuit, into an interval graph. Another similar heuristic approach was presented by Li [15], in which he found the minimal augmentation on a "Vertex-versus-Dominant-Gates" matrix. In addition to this interval graph based approach, Cheng [16] proposed a min-cut algorithm and Hwang *et al.* [12] used a "modified min-net-cut" method to solve this linear placement problem.

(2) Hierarchical approach: This approach was proposed by Yamada *et al.* [17] who used a heuristic algorithm based on hierarchical contraction of nets. The original problem is partitioned into multiple levels in a bottom-up manner by contracting multiterminal nets. Then the gate assignment at each level is determined in a top-down manner.

(3) Branch-and-bound technique: This technique was proposed by Asano [18] to prevent exhaustive search in order to find an optimal net ordering. The gate sequence can be derived from the net ordering in a straightforward manner. By using node denomination, the unnecessary branching can be avoided. The bounding operation is done by comparing the minimum track usage achieved during the search process. It has been proved that optimal solution can always be found using this approach. However, the execution time will grow very fast for large size circuit.

(4) Artificial intelligence approach: Artificial intelligence has growing popularity in various applications. Hu and Chen proposed a planning-based gate matrix layout algorithm, called GM_Plan, which combines the gate placement and net routing into a single problem solving loop [19]. The overall goal of gate matrix layout consists of many subgoals each of which corresponds to the placement of a gate to a slot, and to the routing of associated nets connecting to that gate. As different nets compete for track usage, these subgoals interact with each other rendering suboptimal solutions. The interaction among subgoals is managed with two artificial intelligence planning techniques: hierarchical subgoal organization and meta-planning policies. Two meta-planning policies—*graceful retreat* and *least impact*—are used to guide the search process.

2.2.2 Iterative methods Iterative methods attempt to improve a complete layout by producing a new better one. Constructive layout algorithms are normally used for initial layout, and are usually followed by iterative algorithms. A good constructive initial layout helps the iterative algorithm produce best solution sooner. Iterative algorithms are typically divided into the following classes: (1) Pairwise interchange, (2) Simulated Annealing (SA), (3) Iterative Learning approach, and (4) Genetic algorithm (GA).

(1) Pairwise interchange: There are iterative algorithms that attempt to repeatedly improve gate-matrix layout solutions through pairwise interchange of gates [20, 21].

(2) Simulated Annealing: Simulated Annealing, proposed by Kirkpatrick *et al.* [10], is a general combinatorial optimization technique that uses a stochastic hill-climbing method to search for optimal solutions. The Timerwolf package used in the standard cell placement problem, which is based on SA, has proven to be an effective tool for small- and medium-size designs and has been widely used. Other implementation for gate matrix layout has been reported in [22]. It is noted that such an implementation requires careful tuning of some control parameters to achieve good results [23, 24].

(3) Iterative Learning approach: GM_LEARN [30], is an iterative approach based on GM_Plan. Two A.I. learning technologies, rote learning and learning by parameter adjustment, are employed. The *trapezoid rules* used in GM_LEARN are the simple learning devices incorporating some averaging technique. By repeatedly applying GM-Plan with different weights, better gate-matrix layout can be obtained.

(4) Genetic algorithm: Genetic algorithm was developed by John Holland [7]. Genetic algorithms are search algorithms based on the mechanics of natural selection and natural

genetics. They combine survival of the fittest among string structures with a structured yet randomized information exchange to form a search algorithm. In every generation, a new set of artificial creatures (strings) is created using bits and pieces of the fittest of the old; an occasional new part is tried for good measure. Kotliar and Tabtieng recently reported the application of Genetic algorithm to gate-matrix layout [25].

2.2.3 Gate-matrix partitioning Larger circuits can be laid out by several gate-matrix cells with better aspect ratio and smaller layout area [26]. Huentemann and Baitinger proposed an approach for gate-matrix synthesis by partitioning an EDIF logical network description into gate-matrix "macrocells" for gate-matrix layout generator [27].

3 SIMULATED EVOLUTION GATE-MATRIX LAYOUT ALGORITHM

A block diagram of SEGMA is depicted in Figure 2. The program is written in C, running on SUN SPARC/SLC under SUN OS version 4.1 with 8 Mbyte main memory. A pseudo-code description of the SEGMA algorithm is presented in appendix A. SEGMA begins with a random placement of gates. The nets connecting to the gates are then routed using the routine *Route_nets* which implements the left-edge-first algorithm [28]. This initial layout serves as a starting point for the following iterations (*Evolution*). By generating a new gate sequence from the current generation using a heuristic method to be described below, the quality of the gate-matrix layout is expected to be gradually improved from one generation to the next. The search process will be terminated when predefined termination criteria are satisfied.

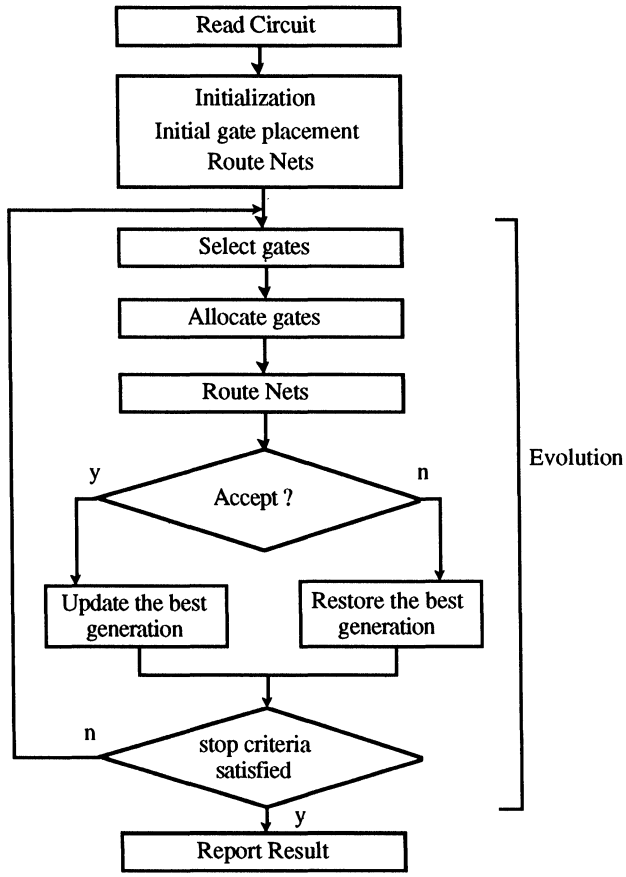


FIGURE 2 Block diagram of SEGMA.

3.1 Selection Criterion and Gate Quality Evaluation

To apply the simulated evolution search strategy, we need to identify a subset of transistor gates and blame them for causing excessive usage of tracks. This is not easy. While the track usage is dictated by the placement of **all** the transistor gates, it is not obvious how they are related with respect to the placement of an **individual** transistor gate. To overcome this difficulty, in SEGMA, we propose to assess the quality of the placement of an individual gate based on net connection length to that gate. The underlying rationale is that if each gate requires minimum net length, the track usage can be reduced as well. To facilitate presentation, we give the following definitions.

Definition 9: The compactness ratio of a net n_i, $V(n_i; f)$ is defined as the ratio of its minimum length over its length $L(n_i; f) \leq n - 1$ under the current gate placement function f. That is

$$V(n_i; f) = \frac{|G(n_i)| - 1}{L(n_i; f)}$$

From the definition above, we have

$$1 \geq V(n_i; f) \geq \frac{|G(n_i)| - 1}{n - 1} \geq \frac{1}{n - 1}. \quad (3.1)$$

A net n_i is compact if $V(n_i; f) = 1$. If a net is less compact (i.e., $V(n_i; f) \to |G(n_i)| - 1/(n - 1)$), it runs over many gates without connecting to them, causing significant waste of track space.

Intuitively, the more compact each net becomes, the more likely fewer tracks will be needed. The compactness of the nets can be used as a criterion to measure the quality of the relative placement of each gate under a placement function f.

Definition 10: The quality factor of a gate g_i, denoted by $Q(g_i; f)$, is defined as the average of compactness ratio of the nets in the set $N(g_i)$. That is,

$$Q(g_i; f) = \frac{\sum_{n_i \in N(g_i)} V(n_i; f)}{|N(g_i)|} \tag{3.2}$$

Substituting (3.1) into (3.2), we have

$$1 \geq Q(g_i; f) \geq \frac{1}{n - 1}. \tag{3.3}$$

The placement of a gate under f is of high quality if all the nets connecting to it are sufficiently compact. Those gates with poor qualities are likely to be in the wrong slots and thus probably should be replaced.

For standard-cell placement problems, a similar definition of the quantity $1 - Q(g_i; f)$ has been utilized in ESP as the probability that each cell may be selected in the evaluation phase. At the beginning, when most of the cells have poor quality, more cells will be selected. Upon convergence, when most of cells have higher quality, fewer cells will be selected. However, the actual number of cells selected is random and cannot be easily predicted for a specific problem. Due to the different nature of the gate-matrix layout problems, a different strategy is developed in SEGMA: Before gates are selected, the quality factor of each gate is added to a random number uniformly distributed over [0, 1]. The sum is defined as the *randomized quality factor* $Q_r(g_i; f)$ of gate g_i. This randomization step serves two purposes: (1) With the randomized quality factor, every gate will have a non-zero probability to be selected for placement. From inequality (3.3), we have

$$\begin{aligned} &\text{prob}\{Q_r(g_i; f) < Q_r(g_j; f) \\ &\quad = \text{prob}\{Q(g_i; f) + r_i < Q(g_j; f) + r_j\} \\ &\quad = Q(g_j; f) - Q(g_i; f) + 1 \geq \frac{1}{n - 1} > 0 \end{aligned}$$

even with $Q(g_i; f) = 1$ for random numbers r_i and r_j uniformly distributed in the range [0, 1]. This will help preventing the solution from trapping into a local minimum. (2) If $Q(g_i; f) < Q(g_j; f)$ then

$$\begin{aligned} &\text{prob}\{Q_r(g_i; f) < Q_r(g_j; f)\} \\ &\quad = 1 + [Q(g_i; f) - Q(g_i; f)] \\ &> \text{prob}\{Q_r(g_i; f) > Q_r(g_j; f)\} \\ &\quad = 1 - [Q(g_j; f) - Q(g_i; f)] \end{aligned}$$

In other words, poor quality gates are more likely to be selected. Thus the randomization strategy used in SEGMA strives to seek a balance between two conflict objectives: seeking globally optimal solution and enhancing search efficiency.

One the randomized quality factor for each transistor gate is evaluated, all the transistor gates will be sorted according to their randomized quality factors in ascending order. A predetermined number of gates (referred to as the *window size* b thereafter) with lower scores will be selected into a queue. The nets originally connected to these b gates will be disconnected. In SEGMA we set window size $b = 0.25 \times n$, where n is the total number of gates. Previously, a random window size was used in both ESP [3] and SILK [6] as the part of selection criteria. Although ours is a fixed window size, the randomization of the quality factor guarantees that every gate has a chance to be selected provided $b \geq 2$. We have also investigated the strategy of the decreasing and expanding window size as the iteration progresses. According to our experience, the decreasing window size strategy performs similarly to the fixed window size strategy except the circuit "w1" in the gate-matrix layout benchmark problems we have tested. Compared with these two strategies, an expanding window strategy often yields worse results.

3.2 Allocation Strategy

After the lower-quality gates are evicted, the remaining gates will stay in their assigned slots with the b emptied slots interlaced between them. The selected gates then will be redeployed back to these slots one by one, starting from the gate with the lowest score.

To place a gate g_w, we calculate, for each emptied slot, an *estimated quality factor* $\hat{Q}(g_w; f)$ when g_w is placed into that slot. This is an *estimated* quality factor because not all gates have been placed. Certain nets may have not been connected to all the gates specified in their gate set $G(n_i)$ after the selection process. Some nets may even be completely missing

due to the eviction of gates which they connect to. Therefore $\hat{Q}(g_w; f) \neq Q(g_w; f)$ where $Q(g_w; f)$ is the original quality factor before randomization.

After all the emptied slots have been tried, the slot which yields the maximum estimated quality factor $\hat{Q}(g_w; f)$ will be designated for placing g_w, and the nets having contacts with g_w, will be reconnected. Then we proceed to place the next gates in the queue. This process is repeated until all the gates in the queue are placed. Then the left-edge-first algorithm will be applied to route all the nets. Thus we have a new design.

Note that the allocation strategy makes use of a greedy heuristic which always places a gate into a slot with maximum estimated quality factor. An alternative approach would be to randomize the estimated quality factor by adding it to a random number uniformly distributed between 0 and 1, just like the randomized quality factor used during the selection process. We choose the greedy heuristic approach over the randomized approach for the benefit of performance. A simulation comparing these two schemes is given in section V.

3.3 Acceptance Criterion for Evolving into a New Generation

After a new design is generated, we must decide whether to evolve into the new design. In SEGMA a new layout will replace the current best layout (current generation) if it uses equal or fewer tracks than the current layout. Otherwise this new design will be discarded and the current generation remains the same.

Although inherently this acceptance strategy is a greedy heuristic algorithm, it is, in fact, not quite so. To illustrate this, we have conducted an experiment using exhaustive searching to find all the possible gate-matrix layouts of the circuit shown in Figure 3(a) (7! = 5040 in this case). The frequency distribution of layouts with specific total net length and track number is plotted in Figure 3(b). Also, the conditional probability of the optimal solution (track number = 5) for a given total net length is depicted in Figure 3(c). From this figure, two observations can be made: a) Layouts with the same track number have a large variation in term of net lengths. b) The smaller the total net length, the higher the probability that the track number will also be smaller. Although we only accept layout with equal or fewer tracks, those layouts may still use longer total wire lengths. Since the quality factor is defined on the compactness of the wire length, a new design having longer wire length (even use equal or fewer tracks) can be regarded as one having inferior quality. Again, as we have repeatedly emphasized, this accepting strategy represent a compromise between the two conflicting goals of searching for globally optimal solution and enhancing search efficiency.

3.4 Termination Criteria

Two termination criteria are devised to determine whether to terminate the evolution iterations:

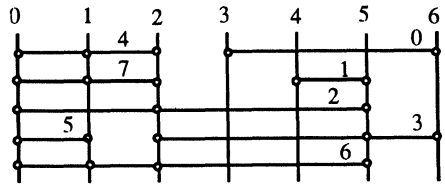
1) Terminate the evolution iteration if the lower bound of the track number (cf. equation (2.1)) is attained. The maximum number of nets connecting to a single gate is a lower bound (not necessarily always the tightest) of the track number. When this lower bound is reached, an optimal solution is obtained. Note that for some circuits the globally optimal solution may need more tracks than this lower bound. In this case, this termination criterion will never be satisfied.
2) Terminate the evolution iteration if the circuit has not been improved for $R = 20 \times n$ consecutive iterations. A variable c is used to help determine if this criterion is met. Initially $c = R$. When a new design fails to be accepted as a new generation, c is decreased by 1. Otherwise, $c = c + R$. This strategy is based on the heuristic that an improvement in the evolutionary course is a good indication that an improvement is likely to occur soon.

If neither of these criteria is satisfied, the next evolution iteration will commence. When a new design fails to be accepted as a new generation in the previous iteration, we must perform the gate selection from the same generation of gate-matrix layout as in the previous iteration. Note that the randomization of the quality factor makes it almost impossible to select exactly the same set of gates in successive iterations. To avoid excessive computing time, a cop on the total executing time or the total number of iterations can be also incorporated.

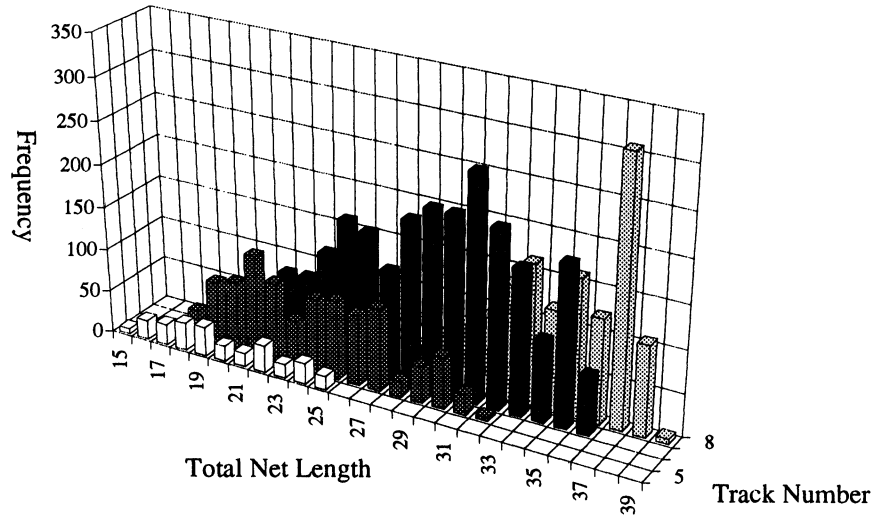
4 COMPLEXITY ANALYSES, BENCHMARK RESULTS, AND CONVERGENCE

4.1 Complexity Analysis

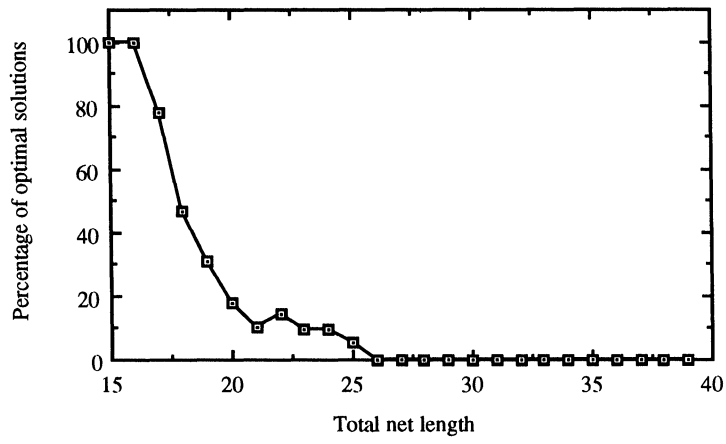
Several notations will be used in the following analysis. $\beta = \max_{g_i \in G} |N(g_i)|$. $\gamma = \max_{n_i \in N} |G(n_i|$. The



(a)



(b)



(c)

FIGURE 3 The relation between total net length and track usage. (a) The gate-matrix layout of the circuit. (b) The frequency distribution of layouts with specific total net length and track number. (c) The conditional probability of the optimal solution for a given total net length.

TABLE I
Comparison of Track Usage

Circuit	No. of gates $\|G\|$	No. of nets $\|N\|$	Results of others $H(f)$	SEGMA: Best of 5 runs $H(f)$	SEGMA: Best of 5 runs Time (sec)	SEGMA: Worst of 5 runs $H(f)$	SEGMA: Worst of 5 runs Time (sec)
wan [31]	7	8	*6 [31]	*6	0.02	*6	0.08
wli [15]	10	11	*4 [15]	*4	0.05	*4	0.72
w1 [14]	21	18	*4 [14]	*4	2.48	6	0.45
w2p [19]	33	39	15 [19]	*14	1.55	*14	3.98
v4470 [29]	47	37	11 [30]	9	128.42	11	71.55
x0 [32]	48	40	13 [30]	11	85.06	13	2.87
w3 [31]	70	84	21 [19]	20	2760.22	24	297.70
w3n [19]	70	130	41 [19]	28	1425.64	31	2001.80
w3p [19]	70	130	32 [19]	27	1865.71	31	1437.20
no8 [17]	85	96	23 [17]	20	322.08	22	206.90
w4 [31]	141	202	34 [30]	30	1243.30	37	2537.40

(*optimal solution)

track usage in a gate-matrix layout is denoted by t. The maximum number of selected gates is k. Obviously, $n \geq \gamma$, $m \geq t \geq \beta$, and $k = n/2$.

Since the number of iterations is hard to predict and is determined by (1) the complexity of the circuits, (2) the effectiveness of the evolution process, (3) the initial condition, and (4) the termination criteria. Thus, our analysis will focus on the complexity analysis of each iteration. That is, the complexity of the *while loop* in the routine *Evolution* in Appendix A.

Refer to the *while loop* in the *Evolution* routine. The routine *Select_gates* and *Sort_gates* are merged into one routine in our implementation. In this routine, to calculate the quality factors of the gates takes $O(n\beta)$ time units. To sort the gates in the queue takes $O(n \log n)$ time units. To place gates in the routine *Allocate_gates* takes $O(n^2)$ time units. To route nets in the routine *Route_nets* takes $O(n\beta t)$ time units. The routine *Update_gate_sequence* and *Restore_gate_sequence* take $O(m)$ time units. Of all the routines, it seems that the routine *Route_nets* that takes $O(n\beta t)$ time units is the most complicated one. However, as mentioned before, β is a small constant in most cases and t grows more slowly than n. $O(n^2)$ should be larger than $O(n\beta t)$ in most cases. Therefore we conclude that the complexity of an iteration in SEGMA is $O(n^2)$ where n is the number of gates.

4.2 Benchmark Tests

Eleven circuits have been tested to compare SEGMA with existing known gate-matrix layout algorithms. The objective is to verify that SEGMA is capable of finding the best design in a reasonable amount of time. The results are listed in Table I. The entries in the "circuit" column give the names and references of the corresponding circuits. The entries in the "Results of others $H(f)$" column give the best known results to date. The entries in the "*Best of 5 runs of SEGMA* $H(f)$" and "*Worst 5 runs of SEGMA* $H(f)$" are taken as the best and the worst results out of five runs of each circuit. The time is measured in seconds.

Note that SEGMA consistently produces the best results for all the 11 circuits. As an illustration, the circuit "w2p" shown in Figure 4 is optimal because the track usage reaches its lower bound.

4.3 Comparison with the Simulated Annealing Method

As mentioned earlier in this paper, simulated evolution and simulation annealing are both stochastic search algorithms with different search strategies. Hence, it would be instructive to compare the performance of SEGMA with a simulated annealing based gate-matrix layout algorithm in terms of both convergence rate and final track usage.

We chose to use a single-loop *SA* algorithm rather than a double loop implementation as the latter performs poorly compared with the single loop formulation. A generic pseudo-code listing of the single loop SA algorithm is given below:

```
Simulated_Annealing (i0, T0) {
T = T0;                          /* T0 is initial temperature */
i = i0;                          /* i0 is an initial state */
while (stopping criterion is not satisfied) {
  j = generate (i);              /* j is a generated new state */
```

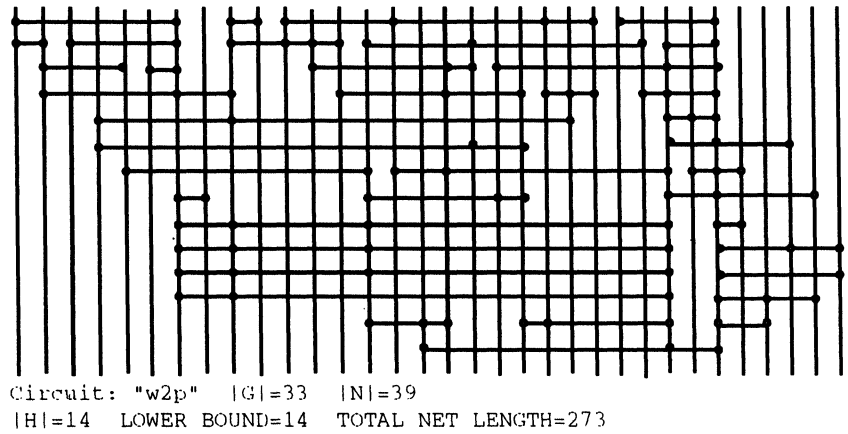


FIGURE 4 Optimal layout result of circuit "w2p."

```
    if (accept(c(i), c(j), T)) i = j;   /* judge the acceptance of state j */
    T = update (T);                     /* update temperature */
  }
}
accept(c(i), c(j), T) {
  Δc = c(j) − c(i);
  y = a(Δc, T)
  r = random(0, 1);
  if (r < y) return(TRUE);
  else return(FALSE);
}
```

The SA based gate-matrix layout algorithm has been carefully fine-tuned to produce the best result. In the module *generate,* we interchange the placement of two gates with lowest quality factors to generate a new state. Assuming f_1 is the mapping function for state i and f_2 for state j, the cost function $c(i) = H(f_1)$ and $c(j) = H(f_2)$. In the *accept* function, $a(\Delta c, T) = 1$, if $\Delta c \leq 0$, otherwise $a(\Delta c, T) = e^{(-\Delta c/T)}$. If y is greater than a random number ranging from 0 to 1, the accept function return TRUE (FALSE otherwise). In function *update,* $T = e^{(-\beta T/T_0)}$ where $\beta = 0.9$. The initial temperature is calculated as $T_0 = |N| - \max\{|N(g_i)|\}$. The initial state i_0 is randomly assigned. The stopping criteria are: (1) the temperature T is reduced to final temperature T_f or (2) the optimal solution is reached.

Because the first stopping criteria of *SE* and *SA* are different, only the first stopping criterion which terminates the iterations when the track number used is equal to the lower bound of the circuit, is used here for fair comparison. Four circuits, whose lower bound are known, are used in this experiment. The results are illustrated in Table II. Iterations, execution time, and the percentage of the optimal solutions the algorithms can produce are recorded for *SE* and *SA*. The iteration and execution time are only the averages of producing twenty optimal solutions. A speedup defined as time of *SA* over time of *SE* is also calculated. If counting the capability to produce optimal solutions, the *SE* will outperform *SA* more than the results listed in Table II.

To illustrate the relationship between deterministic method, SEGMA, and simulated annealing method, a further experiment is introduced. A large circuit "no8" is laid out by three kinds of available algorithms: deterministic methods GM_Plan [19] and net contraction method [17], the simulated evolution based method SEGMA, and the *SA* method with $\beta = 0.99$. The distribution of track number

TABLE II
Benchmark Results for Number of Iterations

Circuit	Iterations of *SA*	Time (s) of *SA*	% of Optima	Iterations of *SE*	Time (s) of *SE*	% of Optima	Speedup
w1	10074	67.92	12.5	284	3.33	87.5	204.9
w2p	1021	25.84	100	125	4.96	100	5.21
wli	148	0.46	40	53	0.28	95	1.78
wan	21	0.052	100	10	0.038	100	1.37

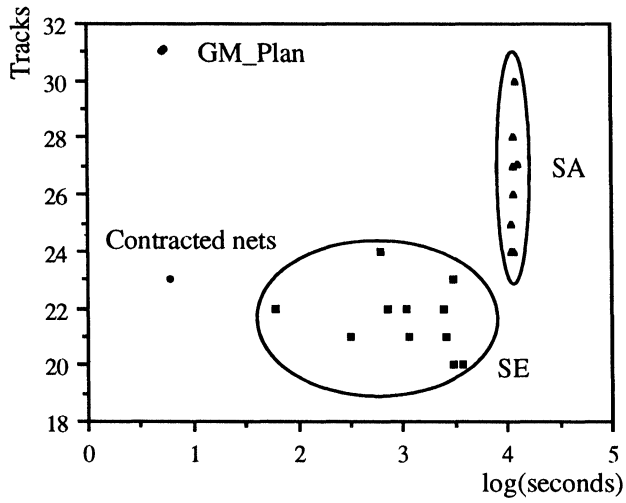


FIGURE 5 The track usage of the circuit "no8" by using *SE*, *SA*, and deterministic methods.

versus iteration time is depicted in Figure 5. We observe that deterministic methods can generate a layout very fast but the layout quality is not always the best, and cannot be easily improved. *SA* will find better layouts given sufficiently long search time. But its convergence rate is too slow to be practical for many applications. SEGMA seems to offer the best of these two extremes as it produces best results at moderate cost of computing time.

In Figure 5, the circuit "no8" is tested ten times by using *SE* and *SA* respectively. *SA* can layout the circuit by using fewer tracks than GM_Plan in expense of the execution time. Impressively, *SE* in this test outperforms *SA* both in the layout quality and execution time. It is proved that *SA* can always find optimal solution if the execution time is long enough. In this case, *SA* can layout the circuit with the same quality with *SE* by taking more time. Note that the time restriction for SA is 10^4 seconds. The contracted-net method is a special design for the circuit "no8." This result cannot be treated as the average one generated by deterministic methods.

4.4 Convergence Analysis

Being an iterative algorithm, it is necessary to discuss the convergence properties of the SEGMA algorithm. In this section, we will establish conditions under which SEGMA will converge to the globally optimal solution in probability when infinite number of iterations are allowed.

Kling *et al.* [4] has established the global convergence of the SE algorithm by modeling the evolution process as a Markov chain. We would like to argue that the global convergence of SEGMA can also be established similarly with some modification of the currently implemented search strategy. For this purpose, we regard each particular gate placement as a state in Markov chain. The transition from one state to the other is done in two subsequent places: First, b (≥ 2) gates are selected. Next, these selected gates will be reassigned to the b emptied slots. The new state will differ from the old state in at most b gate positions. Thus, each state is able to make transition into neighboring states with *non-zero probability*. One may show that such a state transition makes up an *ergodic Markov chain* in which the probability to make a sequence of transitions from any initial state to any of the globally optimal states is greater than zero. Therefore, the global convergence can be established.

The key to prove the global convergence is that state transition probability to the neighboring states are non-zero. The current formulation of SEGMA does not always guarantee this condition. In particular, the use of deterministic estimated quality factor to perform allocation, and the criteria to accept only equal or better new design as the new generation may prevent the transition to some neighboring states from the current state. Therefore, strictly speaking, SEGMA is not guaranteed to converge to the globally optimal solution. On the other hand, by introducing randomization during the selection phase, SEGMA exhibits excellent layout results compared with those obtained from the other greedy algorithms.

5 EXPERIMENTAL STUDIES OF DIFFERENT IMPLEMENTATION STRATEGIES

5.1 Initial Gate Placement

In this section, we would like to investigate the effects of the quality of the initial gate-matrix layout on (a) the quality of the final gate-matrix layout and (b) the convergence rate of SEGMA. We use the circuit "w1" as an example. In this experiment, we use six different initial conditions where five of the initial track numbers are equal to 5, 6, 8, 10, 13 respectively. The sixth has its initial track number randomly chosen from the set of 5, 6, 8, 10,13. For each of the six different initial conditions, 100 independent runs of SEGMA are performed, and the corresponding execution time and final track number are recorded. The results are summarized in Figure

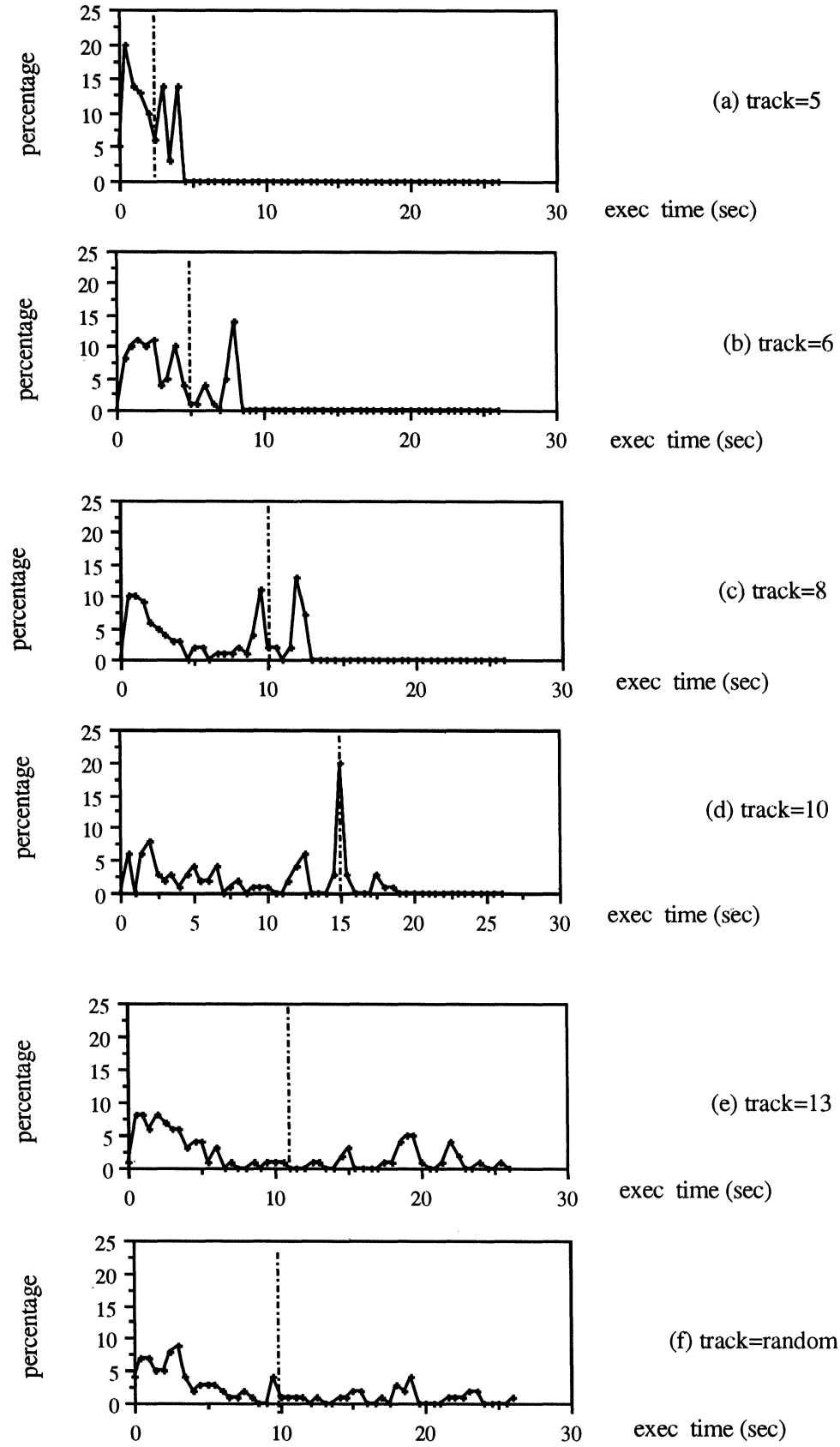


FIGURE 6 Distributions of execution time versus initial track numbers.

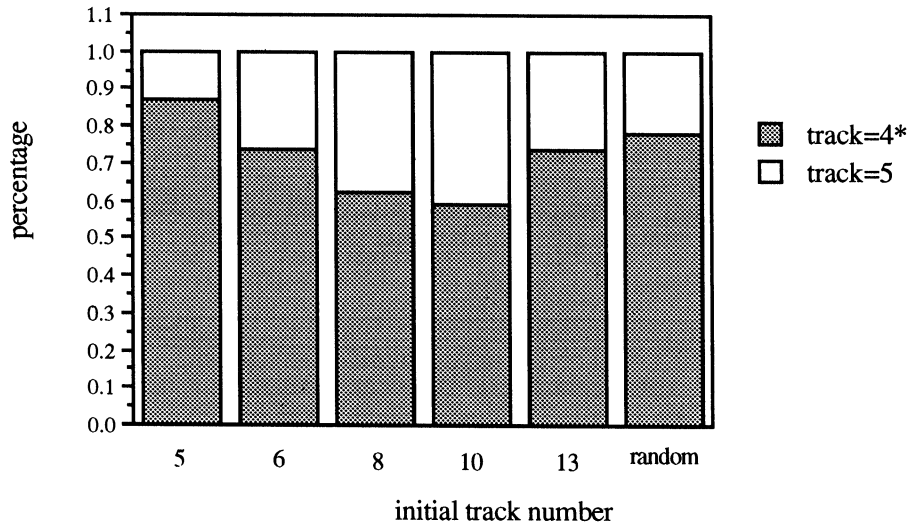


FIGURE 7 Distribution of final track number versus initial track number.

6 (a) to (f). In these Figures, the distribution of execution time (measured in seconds) is plotted for each of the six initial conditions. The dotted vertical line is the average execution time for the given initial condition. Clearly, better initial layouts (fewer initial tracks) yield faster convergence of SEGMA. However, since generally it is difficult to assess how good an initial gate-matrix layout is, using a random initial gate placement is not likely to significantly prolong the convergence.

In Figure 7, the distribution of the final track number versus the initial track number is plotted. It turns out that all simulation runs return a final track number of 4 or 5 in this example. Generally speaking, when the initial condition is sufficiently close to the optimal one, there seems to be a higher probability that the result will also be optimal. Again, on average, the random initial condition strategy adopted in SEGMA will be comparable to the average performance when the initial conditions are specifically selected. Same experiments performed on other circuits yields similar results. Thus we conclude that unless sufficient evidence indicates that the initial condition is close to the optimal one, the random initial condition used in SEGMA should be satisfactory.

5.2 Deterministic vs. Randomized Allocation Strategy

In section 3.2, we mentioned that randomized allocation strategy can be implemented by randomizing the *estimated quality factors*.

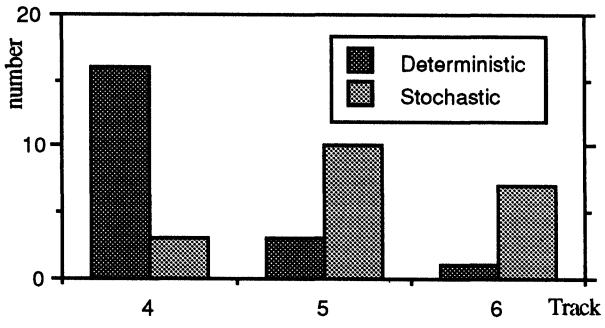
To compare the deterministic allocation strategy with the stochastic allocation strategy, we conduct the following experiment: We choose three circuits "w1," 'v4470," and "no8," and repeat the experiment 20 times for each circuit with each of the two allocation strategies. The results are summarized in Figure 8 in which the distribution of the final track numbers of these two allocation methods are plotted. Clearly, the deterministic allocation strategy outperforms the stochastic allocation strategy by a wide margin. This observation justifies our decision to bias toward the deterministic allocation strategy.

5.3 Comparison of the Two Acceptance Strategies to Accept a New Design

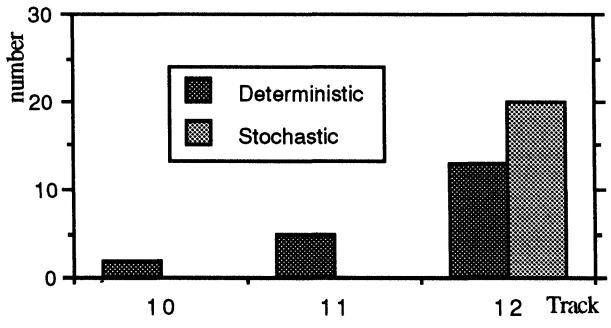
Our objective is to compare the effectiveness of the two acceptance strategies—that is, accepting a new design only when it is *better than* the current generation versus accepting a new design as long as it is as good as the current generation. We have selected three circuits, "w1" (small), "v4470" (middle), and "no8" (large), to test. Each strategy has been tried twenty times on each of these three circuits. The distribution of the final track usage is summarized in Figures 9(a), (b), and (c). We observe that for these three circuits the equal or better acceptance strategy used in SEGMA leads to better results for the gate-matrix layout problem.

6 CONCLUSIONS

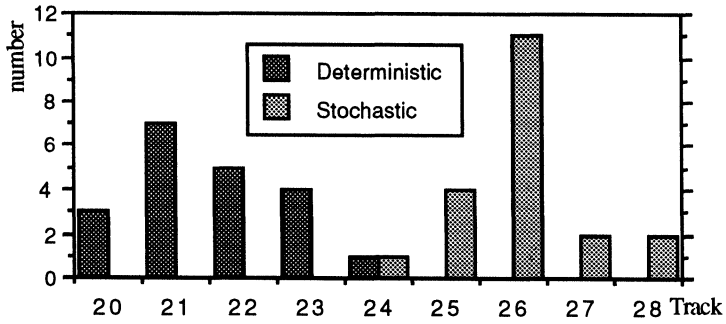
A novel iterative gate-matrix layout algorithm based on a simulated evolution approach has been pro-



(a) Circuit "w1", $|G| = 21$, $|N| = 18$

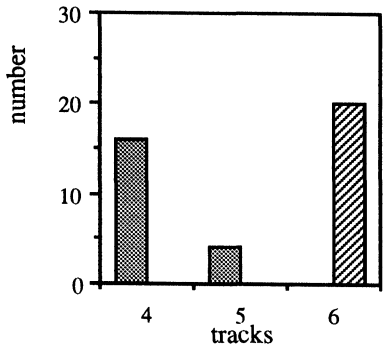


(b) Circuit "v4470", $|G| = 47$, $|N| = 37$

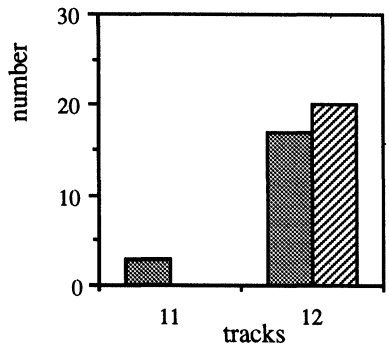


(c) Circuit "no8", $|G| = 85$, $|N| = 96$

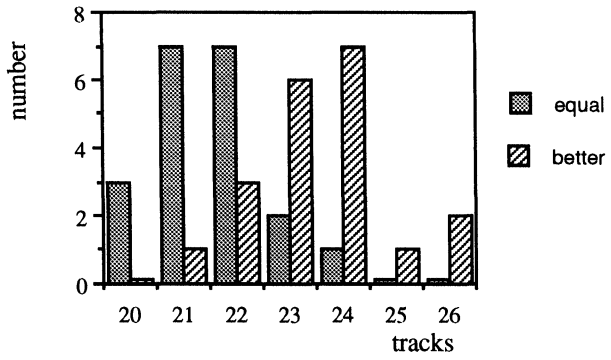
FIGURE 8 The comparison of deterministic and stochastic allocation.



(a) Circuit "w1"



(b) Circuit "v4470"



(c) Circuit "no8"

FIGURE 9 The comparisons of two acceptance strategies.

posed in this paper. Specifically,

- we described the implementation details of the package SEGMA.
- we proposed a novel heuristic criterion to identify individual transistor gates whose placement locations are inferior. This criterion enables the application of the simulated evolution method.
- we developed several effective implementation strategies which leads to very satisfied results as verified with benchmark test.

APPENDIX A

The routines of SEGMA are listed as follows:

```
Main () {
  Read a circuit;
  Initialization ();
  Place gates randomly;
  Route_nets ();
  Evolution ();
  Report results;
}

Initialization () {
  n = number of gates;
  m = number of nets;
  b = 0.25 × n;
  R = 20 × n;
  lower_bound = max (contact net number of gates);
}

Route_nets () {      /* left-edge-first algorithm */
  set T_occupied_k FALSE for each track t_k;
  for each slot s_i occupied by gate g_i from left to right {
    for each net n_j connected to gate g_i {
      if (net n_j has not been routed) {
        choose a track t_k whose T_occupied_k = = FALSE;
        route net n_j on track t_k;
        T_occupied_k = TRUE;
      }
      else if (net n_j has been routed already on on track t_k) {
        T_occupied_k = FALSE;
      }
    }
  }
  calculate track_usage;
  calculate compactness ratio for each net;
}

Evolution () {
  c = R;
  best_result = track_usage;
  while ((c > 0) or (track_usage > lower_bound)) {
    Select_gates ();
    Sort_gates ();
```

```
    Allocate_gates ();
    Route_nets ();
    if (track_usage < best_result) {
      best_result = track_usage;
      Update_gate_sequence ();
      c = c + R;
    }
    else if (track_usage = best_result) {
      best_result = track_usage;
      Update_gate_sequence ();
      c = c - 1;
    }
    else if (track usage > best_result) {
    Restore_gate_sequence ();
    c = c - 1;
  }
}

Select_gates () {
  for each gate g_i calculate Q_r(g_i; f);
  move b gates with largest Q_r(g_i; f) to a queue.
  sort the gates in the queue with the key Q(g_i; f).
}

Allocate_gates () {
  for ith g_i in queue {                    /* i = 1 to b */
    for jth slot s_j calculate Q_h(g_i; s_j);    /* j = 1 to b - i + 1 */
    place g_i in slot s_j* in which Q_h(g_i; s_j*) is maximum;
  }
}
```

References

[1] A.D. Lopez and H.-F.S. Law, "A dense gate-matrix layout method for MOS VLSI," *IEEE Trans. Electron Devices,* vol. ED-27, Aug. 1980, pp. 1671–1675.

[2] Y.C. Chang, S.C. Chang and L.H. Hsu, "Automated layout generation using gate-matrix approach," in *Proc. 24th Design Automation Conf.,* 1987, pp. 552–558.

[3] R.M. Kling and P. Banerjee, "ESP: Placement by simulated evolution," *IEEE Trans. CAD,* vol. 8, no. 3, March 1989, pp. 245–256.

[4] R.M. Kling and P. Banerjee, "Optimization by simulated evolution with applications to standard cell placement," in *Proc. 27th Design Automation Conf.,* 1990, pp. 20–25.

[5] Y. Saab and V. Rao, "An evolution-based approach to partitioning ASIC systems," *Proc. 26th Design Automation Conf.,* 1989, pp. 767–770.

[6] Y.L. Lin, Y.C. Hsu and F.H.S. Tsai, "SILK: A simulated evolution router," *IEEE Trans. CAD,* vol. 8, no. 10, Oct. 1989, pp. 1108–1114.

[7] J.H. Holland, *Adaptation in neural and artificial systems,* Ann Arbor, MI: University of Michigan Press, 1975.

[8] M.P. Fourman, "Compaction of symbolic layout using genetic algorithms," *Proc. Intl. Conf. on Genetic Algorithms and Their Applications,* 1985, pp. 141–153.

[9] J.P. Cohoon and W.D. Paris, "Genetic placement," *Proc. IEEE Intl. Conf. on CAD,* 1986, pp. 422–425.

[10] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, "Optimization by simulated annealing," *Science,* vol. 220, 1983, pp. 671–680.

[11] O. Wing, S. Huang and R. Wang, "Gate matrix layout," *IEEE Trans. CAD,* vol. CAD-4, no. 3, July 1985, pp. 220–231.

[12] D.K. Hwang, W.K. Fuchs and S.M. Kang, "An efficient approach to gate-matrix layout," *IEEE Trans. CAD,* vol. CAD-6, no. 5, Sept 1987, pp. 802–809.

[13] T. Ohtsuki, H. Mori, E.S. Kuh, T. Kashiwabara and T. Fujisawa, "One-dimensional logic gate assignment and interval graphs," *IEEE Trans. Circuits Syst.,* vol. CAS-26, Sept. 1979, pp. 675–684.

[14] O. Wing, "Automated Gate Matrix Layout," *Proc. Intl. Symp. Circuits Syst.,* 1982, pp. 681–685.

[15] J.T. Li, "Algorithms for gate-matrix layout," *Proc. ISCAS,* 1983, pp. 1013–1016.

[16] C.K. Cheng, "Decomposition algorithm for linear placement and application to VLSI design," *Proc. ISCAS,* 1985, pp. 1047–1050.

[17] S. Yamada, H. Okude and T. Kasai, "A hierarchical algorithm for one-dimensional gate assignment based on con-

traction of nets," *IEEE Trans. CAD,* vol. 8, no. 6, June 1988, pp. 622–629.
[18] T. Asano, "An optimal gate placement algorithm for MOS one-dimensional arrays," *J. Digital Syst.,* vol. VI, 1981, pp. 1–27.
[19] Y.H. Hu and S.J. Chen, "GM_Plan: A gate-matrix layout algorithm based on artificial intelligence planning techniques," *IEEE Trans. CAD,* vol. 9, no. 8, Aug. 1990, pp. 836–845.
[20] H. Yoshizawa, H. Kawanishi and K. Kani, "A heuristic procedure for ordering MOS arrays," *Proc. 12th Design Automation Conf.,* 1975, pp. 384–393.
[21] M.L. Yu and W.J. Kubitz, "A VLSI cell synthesizer with structural constrain considerations," *Proc. Intl. Conf. Computer-Aided Design,* 1985, pp. 58–60.
[22] H.W. Leong, "A new algorithm for gate-matrix layout," *Proc. Intl. Conf. Computer-Aided Design,* 1986, pp. 316–319.
[23] C. Sechen and A. Sangiovanni-Vincentelli, "The TimberWolf placement and routing package," *Proc. Custom Integrated Circuits Conf.,* May 1984, pp. 552–527.
[24] S. Nahar, S. Sahni and E. Shragowitz, "Experiments with simulated annealing," *Proc. 22nd Design Automation Conf.,* 1985, pp. 748–752.
[25] M. Kotliar and T. Tabtieng, "A genetic gate matrix layout algorithm," *Int. Symposium on Circuits and Systems,* 1992, pp. 2252–2255.
[26] S. Huang and O. Wing, "Gate matrix partitioning," in *Proc. ICCAD,* Nov. 1988, pp. 130–133.
[27] F.H. Huentemann and U.G. Baitinger, "A gate-matrix oriented partitioning approach for multilevel logical networks," in *Proc. EDAC,* Edinburgh, 1990, pp. 327–331.
[28] A. Hashimoto and J. Stevens, "Wire routing by optimizing channel assignment within large apertures," *Proc. 8th annual Design Automation Workshop,* 1971, pp. 155–169.
[29] D.V. Heinbuch, *CMOS3 Cell Library*. Reading, MA, Addison-Wesley, 1988.
[30] S.J. Chen and Y.H. Hu, "GM_LEARN: An iterative learning algorithm for CMOS gate-matrix layout," *IEEE Proc.,* vol. 137, pt. E, no. 4, July 1990, pp. 301–309.
[31] O. Wing and S. Huang, Private correspondence, Jan-Aug. 1988.
[32] K. Nakatani, T. Fujii, T. Kikuno and N. Yoshida, "A heuristic algorithm for gate-matrix layout," in *Proc. ICCAD,* 1986, pp. 324–327.

Biographies

CHI-YU MAO received the B.S. and M.S. degrees in electrical engineering from National Chengkung University, Taiwan. In 1987, he joined Philips Electronics Industries Co. as an electronics engineer. Since 1988 he has been working towards a doctoral degree in electrical engineering at the University of Wisconsin-Madison. His current research interests include VLSI system design methodologies and algorithms.

YU HEN HU received his BSEE degree from National Taiwan University, Taipei, Taiwan, ROC in 1976. He received the MSEE and Ph.D. degrees in Electrical Engineering from University of Southern California, Los Angeles, California in 1980 and 1982, respectively. From 1983 to 1987, he has been an assistant professor of the Electrical Engineering Department of Southern Methodist University, Dallas, Texas. He joined the Department of Electrical and Computer Engineering, University of Wisconsin, Madison, Wisconsin in 1987 as an assistant professor (1987–1989), and is currently an associate professor. His research interests include VLSI signal processing, artificial neural networks, fast algorithms and parallel computer architectures, nonlinear optimization, and computer aided design tools for VLSI. He has published more than 100 journal and conference papers in these areas. He is a former associate editor (1988–1990) for the *IEEE Transaction of Acoustic, Speech, and Signal Processing* in the areas of system identification and fast algorithms.

Dr. Hu is a member of IEEE, SIAM, and Eta Kappa Nu.