A Memetic Algorithm for VLSI Floorplanning

Maolin Tang, *Member, IEEE*, and Xin Yao, *Fellow, IEEE*

***Abstract*—Floorplanning is an important problem in very large scale integrated-circuit (VLSI) design automation as it determines the performance, size, yield, and reliability of VLSI chips. From the computational point of view, VLSI floorplanning is an NP-hard problem. In this paper, a memetic algorithm (MA) for a nonslicing and hard-module VLSI floorplanning problem is presented. This MA is a hybrid genetic algorithm that uses an effective genetic search method to explore the search space and an efficient local search method to exploit information in the search region. The exploration and exploitation are balanced by a novel bias search strategy. The MA has been implemented and tested on popular benchmark problems. Experimental results show that the MA can quickly produce optimal or nearly optimal solutions for all the tested benchmark problems.**

***Index Terms*—Floorplanning, genetic algorithm (GA), local search, memetic algorithm (MA), very large scale integrated circuit (VLSI).**

I. Introduction

FLOORPLANNING is important in very large scale integrated-circuit (VLSI) design automation as it determines the performance, size, yield, and reliability of VLSI chips. Given a set of circuit components, or "modules," and a net list specifying interconnections between the modules, the goal of VLSI floorplanning is to find a floorplan for the modules such that no module overlaps with another and the area of the floorplan and the interconnections between the modules are minimized.

The representation of floorplans determines the size of the search space and the complexity of transformation between a representation and its corresponding floorplan. Existing floorplan representations can be classified into two categories, namely: 1) "slicing representation" and 2) "nonslicing representation." It is commonly believed that nonslicing representations can contribute to better results than slicing representations.

One of the most efficient nonslicing representations is the ordered tree (O-tree) representation proposed by Guo *et al.* [1]. The representation not only covers all optimal floorplans but also has a smaller search space. For a VLSI floorplanning problem of n modules, the search space of the representation is $n!c_n$, where

$$c_n = \frac{1}{2n+1}\binom{2n+1}{n}$$

which is significantly smaller than other floorplan representations. In addition, it only takes $O(n)$ to transform between an O-tree representation and its corresponding floorplan. Moreover, the representation gives geometrical relations among modules, which is valuable for identifying meaningful building blocks when designing genetic operators of evolutionary algorithms. Hence, the O-tree representation is adopted in this paper.

Existing search methods for the VLSI floorplanning problem fall into two categories, namely: 1) "local search" and 2) "global search." Generally, the local search methods are efficient. However, they may not be able to produce an optimal or nearly optimal solution sometimes as their search may be trapped in a local region.

A widely used global search method for VLSI floorplanning problems is genetic algorithm (GA). GAs have been successfully applied to solve slicing VLSI floorplanning problems [2]–[4]. For nonslicing VLSI floorplanning, a GA has also been presented [5]. Since the encoding scheme does not capture any topological information of the floorplans, the performance of the GA is not satisfactory. For example, for the two popular benchmark problems, namely: 1) "ami33" and 2) "ami49," the area usage is less than 90%.

In this paper, we present a memetic algorithm (MA) [6] for a nonslicing and hard-module VLSI floorplanning problem. MAs are population-based metaheuristic search methods which are inspired by Darwin's principle of natural selection and Dawkins' notion of meme [7], defined as a unit of information that reproduces itself while people exchange ideas, and they have been successfully applied on many complex problems [8]–[14].

Our MA is a hybrid GA that uses an effective genetic search method to explore the search space and an efficient local search method to exploit information in the search region. The exploration and exploitation are balanced by a novel bias search strategy. The MA has been implemented and tested on popular benchmark problems for nonslicing and hard-module VLSI floorplanning. Experimental results show that the MA can quickly produce optimal or nearly optimal solutions for all the popular benchmark problems.

The remaining paper is organized as follows. Section II is the problem statement. Section III reviews related work. The MA is discussed in detail in Section IV, and empirical studies on the MA are presented in Section V. Finally, we conclude the MA in Section VI.

Manuscript received April 30, 2005; revised December 22, 2005 and March 2, 2006. This work was supported in part by the National Natural Science Foundation of China under Grant 60428202. This paper was recommended by Guest Editor Y. S. Ong.

M. Tang is with Queensland University of Technology, Brisbane 4001, Australia (e-mail: m.tang@qut.edu.au).

X. Yao is with the University of Birmingham, Birmingham B15 2TT, U.K., and also with the Nature Inspired Computation and Applications Laboratory, Department of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China (e-mail: x.yao@bham.ac.uk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMCB.2006.883268

II. Problem Statement

A module m_i is a rectangular block with fixed height h_i and width w_i, $M = \{m_1, m_2, \ldots, m_n\}$ is a set of modules, and N is a net list specifying interconnections between the modules in M.

A floorplan F is an assignment of M onto a plane such that no module overlaps with another. A floorplan has an area cost, i.e., $Area(F)$, which is measured by the area of the smallest rectangle enclosing all the modules and an interconnection cost, i.e., $Wirelength(F)$, which is the total length of the wires fulfilling the interconnections specified by N. To minimize the costs, a module may be rotated 90°. The cost of a floorplan F is defined as follows:

$$cost(F) = w_1 \times \frac{Area(F)}{Area^*} + w_2 \times \frac{Wirelength(F)}{Wirelength^*}. \quad (1)$$

In the above equation, $Area^*$ and $Wirelength^*$ represent the minimal area and the interconnection costs, respectively. Since we do not know their values in practice, estimated values are used. w_1 and w_2 are weights assigned to the area minimization objective and the interconnection minimization objective, respectively, where $0 \le w_1, w_2 \le 1$, and $w_1 + w_2 = 1$. The interconnection cost is the total wire length of all the nets, and the wire length of a net is calculated by the half perimeter of the minimal rectangle enclosing the centers of the modules that have a terminal of the net on it.

Given M and N, the objective of the floorplanning problem is to find a floorplan F such that $cost(F)$ is minimized. It should be pointed out that although the optimization problem has two objectives, multiobjective optimization techniques may not be suitable for it as the two objectives are not equally important and the weights assigned to the two objectives should be controllable by the designer to meet various requirements for the floorplanning problem.

III. Related Work

A. O-Tree Representation

A floorplan with n rectangular modules can be represented in a horizontal (vertical) O-tree of $(n+1)$ nodes, of which n nodes correspond to n modules $m_1, m_2, \ldots, m_n$ and one node corresponds to the left (bottom) boundary of the floorplan [1]. The left (bottom) boundary is a dummy module with zero width (height) placed at $x = 0$ $(y = 0)$. In a horizontal O-tree, there exists a directed edge from module m_i to module m_j if and only if $x_j = x_i + w_i$, where x_i is the x coordinate of the left-bottom position of m_i, x_j is the x coordinate of the left-bottom position of m_j, and w_i is the width of m_i. In a vertical O-tree, there exists a directed edge from module m_i to module m_j if and only if $y_j = y_i + h_i$, where y_i is the y coordinate of the left-bottom position of m_i, y_j is the y coordinate of the left-bottom position of m_j, and h_i is the height of m_i. Fig. 1(a) shows a floorplan and its horizontal O-tree representation.

An O-tree can be encoded in a tuple (T, π), where T is a $2n$-bit string specifying the structure of the O-tree, and π is a permutation of the nodes. For a horizontal O-tree, the tuple is

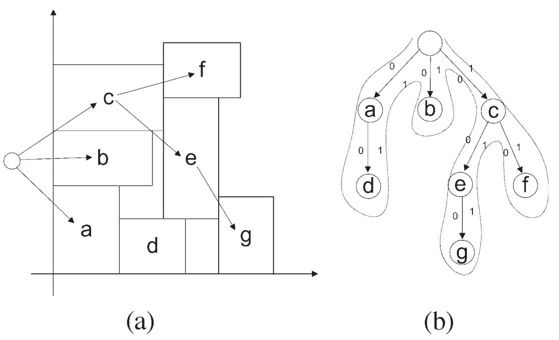

Fig. 1. Illustration of (a) O-tree representation and (b) encoding.

obtained by depth-first traversing the nodes and edges of the O-tree. When visiting a node other than the root, we append the node to π. When visiting an edge in descending direction, we append a 0 to T, and when visiting an edge in ascending direction, we append a 1 to T. The horizontal O-tree shown in Fig. 1(a) is encoded into (00110100011011, $adbcegf$). The idea of the encoding is illustrated in Fig. 1(b). We can use the same idea to encode a vertical O-tree.

A floorplan is "admissible" if no module can be shifted left or bottom without moving other modules in the floorplan. Given a feasible floorplan (a floorplan on which no module overlaps with another), we can derive an admissible floorplan by compacting the modules to the left and bottom boundaries of the floorplan. The cost of the admissible floorplan is equal to or less than that of the original floorplan. Therefore, the search space of the VLSI floorplanning problem is limited to admissible floorplans.

B. Local Search Method

This section presents a local search method for the VLSI floorplanning problem, which is employed by our MA. The local search method is based on a deterministic algorithm proposed by Guo *et al.* [1].

Given an initial floorplan encoded in an O-tree (T, π), the local search method finds a local optimal solution through systematically examining those O-trees that can be obtained by removing a module from, and then putting it back to, the O-tree. When a module is added, its orientation may be changed if it leads to a smaller cost. The algorithm is shown below.

1) For each node m_i in (T, π):
   a) delete m_i from (T, π);
   b) insert m_i in the position where we can get the smallest cost value among all possible insertion positions in (T, π) as an external node of the tree;
   c) perform (a) and (b) on its orthogonal O-tree.
2) Output (T, π).

IV. Our MA

Our MA is a hybrid GA that uses an effective genetic search method to explore the search space and capitalizes on the local search method introduced in the previous section to exploit information in the search region. The exploration and exploitation are balanced by a novel bias search strategy.

A. Genetic Representation

In our MA, each individual in the population is an admissible floorplan represented by an O-tree and encoded in a tuple (T, π), where T is a $2n$-bit string identifying the structure of the O-tree, and π is a permutation of the nodes.

B. Fitness Function

The VLSI floorplanning is a minimization problem, and the objective is to minimize the cost of floorplan F, i.e., $cost(F)$. Thus, the fitness of an individual (T, π) in the population is defined as follows:

$$f((T,\pi)) = \frac{1}{cost\left(F_{(T,\pi)}\right)} \tag{2}$$

where $F_{(T,\pi)}$ is the corresponding floorplan of (T, π), and $cost(F_{(T,\pi)})$ is the cost of F defined in (1).

C. Initial Population

An individual in the initial population is an O-tree (T, π) representing an admissible VLSI floorplan F. A constructive algorithm is designed to construct an admissible O-tree. The algorithm starts with randomly generating a sequence of modules π. Then, it inserts the modules into an initially empty O-tree T in the randomly generated order. When inserting a module into T, it checks all external insertion positions for the module and inserts the module at the position that gives the best fitness. The constructive algorithm is invoked iteratively to generate an initial population of individuals.

D. Genetic Operators

1) Role of the Genetic Operators in the MA: The role that the genetic operators play in our MA is different from that in GAs. In GAs, crossover is used for both exploration and exploitation, and mutation is used for exploration. In our MA, however, the crossover and mutation operators are only used for exploration, or discovering new promising search regions.

The crossover and mutation operators discover new promising search regions by evolving memes, or units of cultural information, in a way analogous to biological evolution. Memes can mutate through, for example, misunderstanding, and two memes can recombine to produce a new meme involving elements of each parent meme.

It is observed that a subtree of the O-tree represents a compact placement of a cluster of modules. Hence, subtrees are used as memes in our MA. The memes are transmitted and evolved through one crossover operator and two mutation operators, which will be discussed in the following.

2) Crossover: Given two parents, both of which are admissible floorplans represented by an O-tree, the crossover operator transmits the significant structural information from two parents to a child. By recombining some significant structural information from two parents, it is hoped that better structural information can be created in the child.

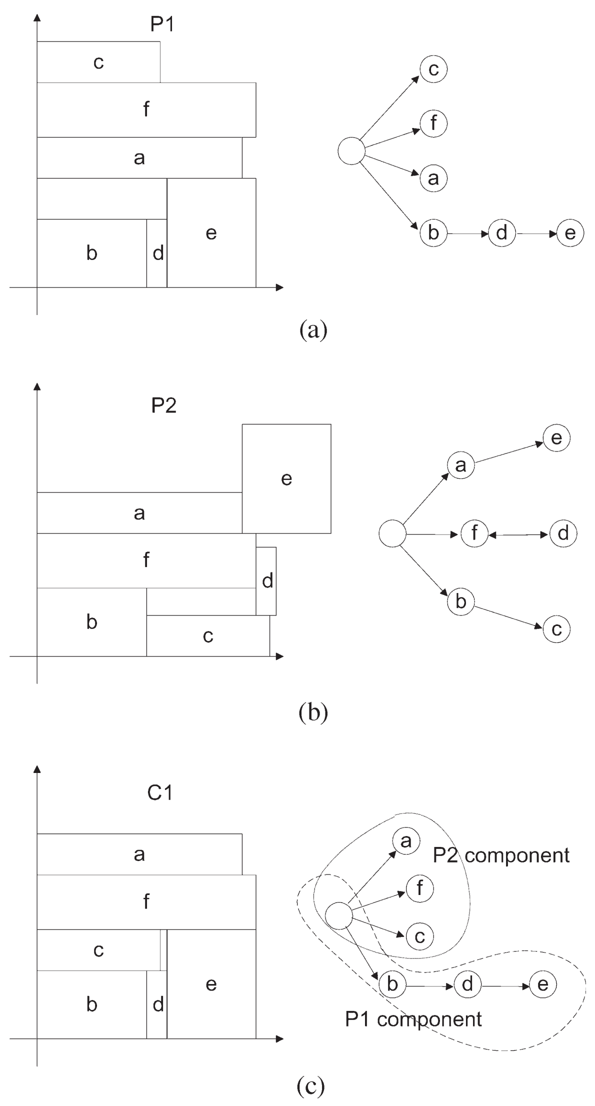

Fig. 2. Crossover operator.

To create a child c_1 from two parents p_1 and p_2, the crossover randomly selects some top-level subtrees from p_1, duplicates them, and puts them in c_1. Then, the crossover operator takes a copy of p_2 and removes those nodes that have been already present in c_1 and then adds the remaining structural components to c_1. In this way, the generated child carries the significant structural information from both p_1 and p_2. Fig. 2 illustrates the basic idea behind the crossover operator. Fig. 2(a) and (b) are two parents, i.e., p_1 and p_2, and Fig. 2(c) is the child produced by the crossover operator. The corresponding admissible floorplans are shown on the left-hand side of the figure.

3) Mutation: Two mutation operators are constructed and used in our MA. The basic idea behind the mutation operators is to discover a new search region by mutating the structure of an individual.

Given an admissible floorplan represented as an O-tree, one mutation operator first identifies the top-level subtrees of the O-tree and then randomly changes the order of the subtrees. This mutation operator is illustrated in Fig. 3, in which Fig. 3(a) shows the initial O-tree and Fig. 3(b) shows the mutated O-tree.

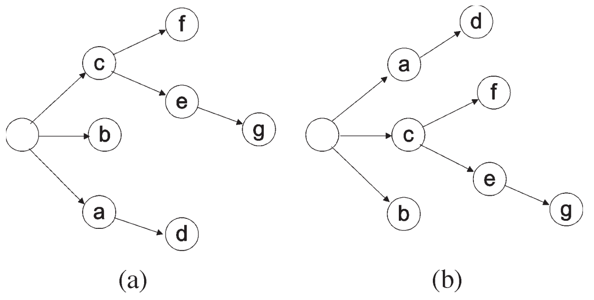

Fig. 3. Mutation operator 1.

Note that the mutated floorplan may not be admissible. Therefore, the mutation operator makes it admissible by moving the modules to the left-hand side and the bottom. When making the mutated floorplan admissible, some of the structural components might be broken and reorganized. However, the mutated floorplan might preserve most of the structural information that the original floorplan has, and therefore, the fitness distance between the mutated and original floorplans would not be long.

A second mutation operator used by our MA randomly selects a subtree at any level, removes it, and then inserts it back to the O-tree. Since the mutated floorplan may not be admissible as in the first mutation operator, it also makes the mutated floorplan admissible in the same way as in the first mutation operator. Fig. 4 illustrates the basic idea behind the mutator. In the figure, (a) shows the initial O-tree and (b) shows the mutated O-tree in which the subtree with root e is being moved.

The two mutation operators are randomly selected and used by our MA to discover new search regions that have different fitness distances from the mutating individual.

E. Strategy to Bias the Search

The search space of the VLSI floorplanning problem is huge, and the number of optima might grow exponentially when the size of the problem increases. Hence, it is impossible for our MA to exploit all the regions. A technique that is usually used to deal with the problem is to use a bias search strategy [15], [16].

In our MA, we use a novel strategy to bias its search. Instead of exploiting all the search points generated by the genetic operators, our MA only exploits those search points (admissible floorplans) whose fitness value is equal to or greater than a threshold v and ignores those search points whose fitness value is less than v. This basic idea behind the bias search strategy is illustrated in Fig. 5.

In Fig. 5, the curve depicts the fitness landscape of a VLSI floorplanning problem. The points o_1 and o_2 are two local optima, and the points p_1, p_2, and p_3 are three search points generated by the genetic operators. The horizontal line is the threshold v. Because the fitness of p_1 and p_2 is less than v, the local search method is not applied on them. The local search method is only applied on p_3.

If the local search method is applied to p_1, the MA would find the optimum o_1, which is remarkably poor. Therefore, by using this bias search strategy, the MA can sort out the search regions where the fitness value of the local optimum is less than v. The MA does not apply the local search method on p_2 because the local search method is computationally expensive, and it takes more time for the local search method to find the optimum o_2 from p_2 than from p_3. (The local search method is a hill-climbing one. Thus, the better starting search points, the shorter the computation time given two starting search points on the same hill.)

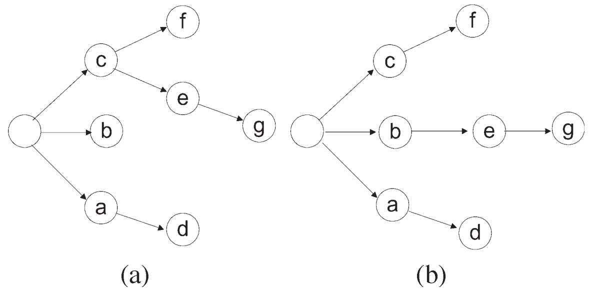

Fig. 4. Mutation operator 2.

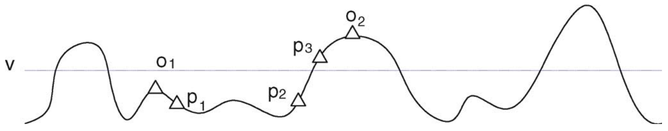

Fig. 5. Strategy to bias the search.

The threshold v is very important. It determines the balance between the exploration and exploitation of our MA. If its value is too big, then some promising search regions may be ignored, and therefore, the chance for our MA to find a global optimum is limited; if its value is too small, then our MA may waste too much time on exploiting less promising search regions, and therefore, the efficiency of our MA is decreased. It is desirable to find a threshold v such that our MA can minimize the use of the local search method without compromising its optimality. An empirical study of the threshold v can be found in Section V.

To test the effectiveness of the threshold strategy, we have compared it with a random search strategy. As its name suggests, the random search strategy randomly picks up individuals for exploitation. Experimental results have shown that the threshold strategy is significant. The details about the comparison can also be found in Section V.

F. Description of Our MA

Initially, the MA randomly generates a population of individuals using the technique described above. Then, the MA starts evolving the population generation by generation. In each generation, the MA uses the genetic operators probabilistically on the individuals in the population to create new promising search points (admissible floorplans) and uses the local search method to optimize them if the fitness of the admissible floorplans is greater than or equal to v. The process is repeated until a preset runtime is up. An outline of the MA is as follows:

1) $t := 0$;
2) generate an initial population $P(t)$ of size $PopSize$;
3) evaluate all individuals in $P(t)$ and find the best individual $best$;

TABLE I
STATISTICS FOR THE TEST RESULTS ON THE PERFORMANCE OF THE MA

MCNC	mDA			GA			MA		
benchmarks	best	mean	std dev	best	mean	std dev	best	mean	std dev
ami33	1225931	1240946	8074	1241219	1278441	14990	1190896	1211159	8155
ami49	37902872	38231446	174441	38069668	38797586	255458	37487940	38093769	228102

TABLE II
KRUSKAL–WALLIS TEST RESULTS FOR THE COMPARISON BETWEEN THE mDA, GA, AND MA

MCNC	mDA			GA			MA			p
benchmarks	n	rank sum	mean rank	n	rank sum	mean rank	n	rank sum	mean rank	
ami33	30	1393.0	46.43	30	2237.0	74.57	30	465.0	15.50	<0.0001
ami49	30	1103.5	36.78	30	2221.0	74.03	30	770.5	25.68	<0.0001

```
4) while the preset runtime is not up:
   a) $t := t + 1$;
   b) for each individual in $P(t)$:
      i) this individual becomes the first parent $p_1$;
      ii) select a second parent using roulette wheel selection $p_2$;
      iii) probabilistically apply crossover to produce a child $c_1$;
      iv) if $fitness(c_1) \geq v$, then optimize $c_1$ using the local search method;
      v) if $fitness(c_1) \geq fitness(p_1)$, then $p_1 := c_1$;
      vi) if $fitness(c_1) \geq fitness(best)$, then $best := c_1$;
      vii) probabilistically apply the two mutators (picked up randomly) on $c_1$ to produce a new individual $f$;
      viii) if $fitness(f) \geq v$, then optimize $f$ using the local search method;
      ix) if $fitness(f) \geq fitness(p_1)$, then $p_1 := f$;
      x) if $fitness(f) \geq fitness(best)$, then $best := f$.
5) output $best$.
```

V. EMPIRICAL STUDIES ON THE MA

In this section, we use two popular benchmarks to empirically study the performance of the MA, the determination of the value of the threshold v, the effectiveness of the threshold bias search strategy, and the effect of weights assigned to the two optimization objectives on the performance of the MA.

The benchmarks used in the empirical studies are two popular MCNC benchmark problems for the VLSI floorplanning problem: ami33 and ami49 [17]. The benchmark ami33 has 33 modules, 123 nets, 480 pins, and 42 input–output (IO) pads. The benchmark ami49 has 49 modules, 408 nets, 931 pins, and 42 IO pads.

A. On the Performance of the MA

To study the performance of the MA, we have implemented the MA in C# on Microsoft Visual Studio .NET 2003. We have also implemented a multistart deterministic algorithm (mDA) and a GA in the same programming language on the same platform for fair comparisons.

The mDA randomly generates search points (admissible floorplans) and optimizes the admissible floorplans using the local search method of the MA. The difference between the mDA and MA lies in their exploration method. The mDA generates search points randomly, whereas the MA generates search points using the genetic operators. Thus, the comparison between the mDA and MA is to justify the effectiveness of the exploration of the MA.

The GA is the MA with the local search method removed. The procedure for generating the initial population, genetic operators, and selection scheme are exactly the same with those used in the MA. Thus, the GA uses the genetic operators for both exploration and exploitation, whereas the MA uses the genetic operators for exploration and uses the local search method for exploitation. Thus, the comparison between the GA and MA is to justify the effectiveness of the exploitation of the MA.

To compare the three algorithms fairly, we run the three algorithms for the same amount of time to see which of the algorithms produces the best result for a test problem. We tested the three algorithms on each the benchmark problems for 30 times, recorded the results obtained from each run, and compared the results obtained by the three algorithms.

For the MA, the population size was set to 10, the probabilities for crossover and mutation were both 0.5, and the threshold v was 0.6 for all the test problems because it was the best configuration found empirically for the MA. For the GA, the population size was set to 100, the probability for crossover was 0.95, and the probability for mutation was 0.05 for all test problems as it was the best configuration found empirically for the GA. We did not use the same configuration for the MA and the GA because it would disadvantage either of them if the other party's best configuration is used. The runtime was 1800 s. In this experiment, the weights for the area minimization and the interconnection minimization objectives were set to 1 and 0, respectively ($w_1 = 1$ and $w_2 = 0$), which means the MA focused on minimizing the area of the floorplan. All experiments were carried out on Pentium IV computers with a 2.0-GHz CPU and a 512-MB RAM. For each of the three algorithms, we recorded the minimal floorplan area obtained in the 30 runs. Table I shows the statistics for the experiment.

It can be seen from Table I that for the benchmark problems, the MA has 1.10% to 2.94% and 0.36% to 2.46% improvement in the best and average solutions, respectively, over the mDA, and has 1.55% to 4.23% and 1.85% to 5.56% improvement in the best and average solutions, respectively, over the GA. To examine the significance of the statistical results, we further performed a Kruskal–Wallis test on the results using a statistical software Analyse-it [18]. The statistical test results are shown in Table II.

To study the dynamic performance of the mDA, GA, and MA, we captured the dynamically changing minimal cost

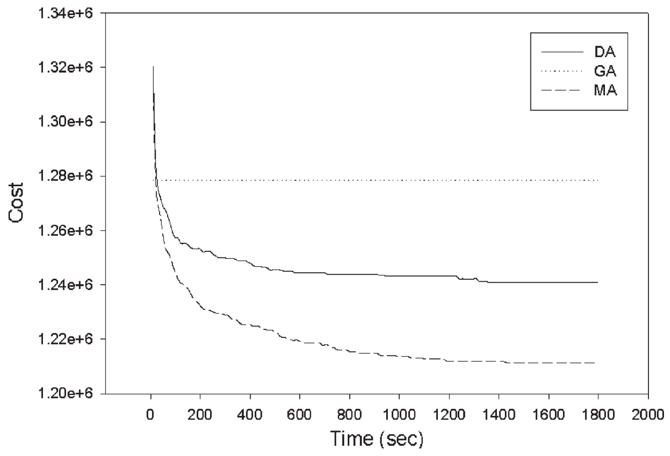

Fig. 6. Dynamic performance comparison between the mDA, GA, and MA on benchmark ami33.

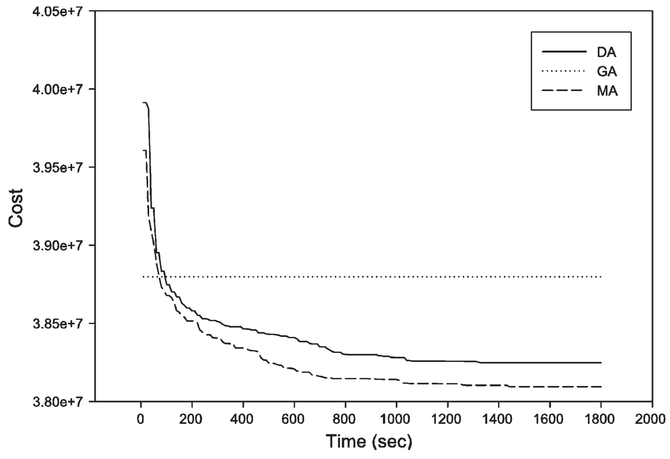

Fig. 7. Dynamic performance comparison between the mDA, GA, and MA on benchmark ami49.

obtained by these algorithms in each run for each of the benchmarks and worked out the average dynamic performance of the three algorithms. Figs. 6 and 7 display the dynamic performance of the mDA, GA, and MA for benchmarks ami33 and ami49, respectively. It can be seen from the figures that the dynamic performance of the algorithms for the two benchmark problems are consistent. The MA is the quickest algorithm that converged to a satisfactory solution of the three algorithms.

B. *On the Determination of the Value of the Threshold v*

The threshold v is very important because it determines the balance between the exploration and exploitation of the MA, which affects the computation time and the optimality of solutions. To determine the value of v, we developed a program to randomly generate 100 search points (admissible floorplans) and used the local search method to optimize them. We applied the program on the two benchmark problems and recorded the fitness values of the initial floorplan and its corresponding optimized floorplan. The experimental results are intuitively shown in Fig. 8. In the figure, the x axis and y axis represent

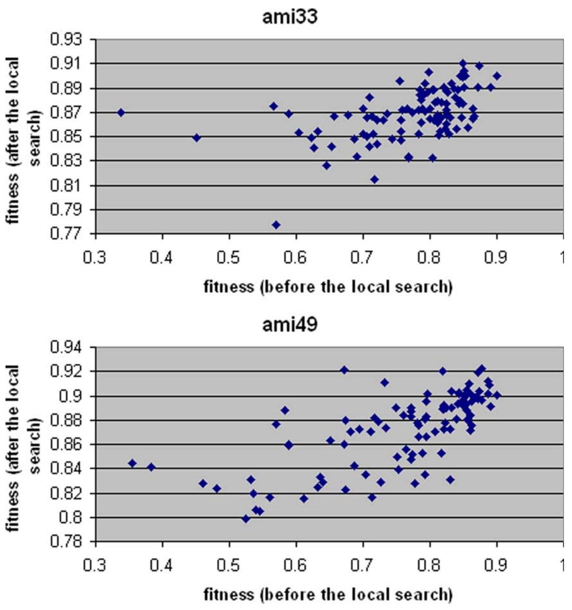

Fig. 8. Determination of the value of the threshold v.

the fitness values of the initial floorplan and its corresponding optimized floorplan, respectively.

There are two observations for all the benchmark problems (Fig. 8).

- The average fitness value of the optimized floorplans from those initial floorplans whose fitness value is less than 0.6 is significantly poorer than that of the optimized floorplans from those initial floorplans whose fitness is equal to or greater than 0.6.
- The best of the 100 optimized floorplans is generated from an initial floorplan whose fitness value is equal to or greater than 0.6.

The two observations suggest that the threshold v should be set to 0.6 for the two benchmark problems. By doing so, the MA may ignore a number of search regions and therefore reduce computation time without compromising the optimality of solutions.

C. *On the Effectiveness of the Threshold Strategy*

To test the effectiveness of the threshold strategy, we developed two MAs, one using the threshold strategy to pick up individuals for local optimization and one randomly picking up 30% of individuals for local optimization (Since there are about 30% of individuals who are selected for local optimization in the threshold strategy, the random selection strategy also picks up 30% of individuals for local optimization for a fair comparison.), and we tested on the benchmark ami33. For each MA, we repeatedly tested on the benchmark problem for ten times. Table III shows the Mann–Whitney statistical test results for the hypothesis "the cost obtained by the MA using the threshold strategy is less than or equal to the cost obtained by the MA using the random picking up strategy." The difference between the medians of the costs obtained by the two MAs is -0.009, and the confidence interval for that is 95.5%.

TABLE III
MANN–WHITNEY TEST RESULTS FOR THE HYPOTHESIS "THE COST OBTAINED BY THE MA USING THE THRESHOLD STRATEGY IS LESS THAN OR EQUAL TO THE COST OBTAINED BY THE MA USING THE RANDOM PICKING UP STRATEGY"

MCNC	The MA with the threshold strategy				The MA with the random selection					
benchmark	n	rank sum	mean rank	U	n	rank sum	mean rank	U	median diff.	p
ami33	10	73.0	7.30	82.0	10	1137.0	13.7	18.0	-0.009	0.0073

TABLE IV
KRUSKAL–WALLIS STATISTICAL TEST RESULTS ON DIFFERENT CONFIGURATIONS

configuration(w_1, w_2)	n	rank sum	mean rank
(1,0)	10	61.0	6.10
(0.5, 0.5)	10	236.0	23.60
(0, 1)	10	168.0	16.80

The statistical test results show that the threshold bias search strategy is significant.

D. On the Tradeoff Between the Two Optimization Objectives

To study the tradeoff between the two optimization objectives, i.e., the minimal area and minimal interconnection, we tested the MA on the benchmark ami33 with three different configurations of weighs, i.e., $\{w_1 = 1,\ w_2 = 0\}$, $\{w_1 = 0.5,\ w_2 = 0.5\}$, and $\{w_1 = 0,\ w_2 = 1\}$. For each of the configurations, we tested the MA on the benchmark for ten times and recorded the test results. Table IV shows the statistical test results.

The statistical test results reveal that the configuration $\{w_1 = 1,\ w_2 = 0\}$ produced the best cost, followed by the configuration $\{w_1 = 0,\ w_2 = 1\}$. The configuration $\{w_1 = 0.5,\ w_2 = 0.5\}$ generated the worst cost.

This paper indicates that the MA is sensitive to the configuration and that the two objectives are mutually constrained with each other. To find the optimal configuration is a tedious job, and the optimal configuration may be different for different problems.

In addition, it has been observed that solutions obtained by the MA generally have smaller area but longer interconnection compared with those obtained by the mDA. Our preliminary research on the crossover operator has revealed that the crossover operator is more favorable to the area minimization objective than the interconnection minimization objective. When recombining two parents to produce a child, the crossover operator can more efficiently preserve those good memes contributing to a compact floorplan, but less efficiently preserve those memes contributing to a short interconnection. Thus, how to improve the crossover operator such that it can effectively preserve good memes contributing to both area and interconnection minimizations is another issue we will investigate in the future.

VI. CONCLUSION

This paper has presented an MA for a nonslicing and hard-module VLSI floorplanning problem—a challenging optimization problem in VLSI design automation. This MA is a hybrid GA that uses an effective genetic search method to explore the search space and an efficient local search method to exploit information in the search region. The exploration and exploitation are balanced by a threshold bias search strategy. Experimental results have shown that the MA outperforms both the mDA and GA for the benchmark problems.

This paper represents our first effort toward efficient MAs for VLSI floorplanning. There are some interesting issues that need to be further investigated.

First, the current MA uses a static threshold bias search strategy—The value of the threshold v is a constant and never changes during the evolution. It is conjectured that if v can dynamically change in the light of the status of the population of the MA, the performance of the MA could be further improved. Therefore, one of the issues that we are going to investigate is dynamic threshold bias search strategies.

ACKNOWLEDGMENT

The authors would like to thank the Associate Editors and anonymous reviewers for their valuable comments. Part of this work was done while the first author was visiting the Center of Excellence for Research in Computational Intelligence and Applications, School of Computer Science, University of Birmingham, U.K.

REFERENCES

[1] P.-N. Guo, T. Takahashi, C.-K. Cheng, and T. Yoshimura, "Floorplanning using a tree representation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 2, pp. 281–289, Feb. 2001.
[2] J. Cohoon, S. Hegde, W. Martin, and D. Richards, "Distributed genetic algorithms for the floorplan design problem," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 10, no. 4, pp. 483–492, Apr. 1991.
[3] M. Rebaudengo and M. Reorda, "GALLO: A genetic algorithm for floorplan area optimization," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 15, no. 8, pp. 943–951, Aug. 1996.
[4] C. Valenzuela and P. Wang, "VLSI placement and area optimization using a genetic algorithm to breed normalized postfix expressions," *IEEE Trans. Evol. Comput.*, vol. 6, no. 4, pp. 390–401, Aug. 2002.
[5] B. Gwee and M. Lim, "A GA with heuristic based decode for IC floorplanning," *Integr., VLSI J.*, vol. 28, no. 2, pp. 157–172, 1999.
[6] P. Moscato, "On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms," California Inst. Technol., Pasadena, Tech. Rep. 826, 1989.
[7] R. Dawkins, *The Selfish Gene*. Oxford, U.K.: Oxford Univ. Press, 1976.
[8] P. Merz and B. Freisleben, "Fitness landscape analysis and memetic algorithms for the quadratic assignment problem," *IEEE Trans. Evol. Comput.*, vol. 4, no. 4, pp. 337–352, Nov. 2000.
[9] E. Burke, P. Cowling, P. Causmaecker, and G. Berghe, "A memetic approach to the nurse rostering problem," *Appl. Intell.*, vol. 15, no. 3, pp. 199–214, Nov. 2001.
[10] A. Quintero and S. Pierre, "A memetic algorithm for assigning cells to switches in cellular mobile networks," *IEEE Commun. Lett.*, vol. 6, no. 11, pp. 484–486, Nov. 2002.
[11] H. Ishibuchi, T. Yoshida, and T. Murata, "Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling," *IEEE Trans. Evol. Comput.*, vol. 7, no. 2, pp. 204–223, Apr. 2003.
[12] K.-H. Liang, X. Yao, and C. Newton, "Larmarckian evolution in global optimization," in *Proc. 26th Int. Conf. Ind. Electron., Control and Instrum. and 3rd Asia-Pacific Conf. Simul. Evol. and Learning*, 2003, pp. 331–334.

[13] Y. S. Ong and A. Keane, "Meta-Lamarckian learning in memetic algorithms," *IEEE Trans. Evol. Comput.*, vol. 8, no. 2, pp. 99–110, Apr. 2004.
[14] N. Krasnogor and J. Smith, "A tutorial for competent memetic algorithms: Model, taxonomy, and design issues," *IEEE Trans. Evol. Comput.*, vol. 9, no. 5, pp. 474–488, Oct. 2005.
[15] H. Ishibuchi and T. Murata, "A multi-objective genetic local search algorithm and its application to flowshop scheduling," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 28, no. 3, pp. 392–403, Aug. 1998.
[16] K.-H. Liang, X. Yao, and C. Newton, "Evolutionary search of approximated n-dimensional landscapes," *Int. J. Knowl.-Based Intell. Eng. Syst.*, vol. 4, no. 3, pp. 172–183, Jul. 2000.
[17] *The MCNC Benchmark Problems for VLSI Floorplanning*. [Online]. Available: http://www.mcnc.org
[18] *The Analyse-it General Statistics Software for Microsoft Excel*. [Online]. Available: http://www.analyse-it.com

Maolin Tang (M'04) received the B.S. degree in computer science from Huazhong University of Science and Technology, Wuhan, China, the M.S. degree in computer science from Chongqing University, Chongqing, China, and the Ph.D. degree in computer systems engineering from Edith Cowan University, Perth, W.A., Australia.

He is currently a Senior Lecturer with the Faculty of Information Technology, Queensland University of Technology, Brisbane, Qld., Australia. His research interests include evolutionary computation, evolutionary VLSI physical design and optimization, and web-based computation.

Xin Yao (M'91–SM'96–F'03) received the B.Sc. degree from the University of Science and Technology of China (USTC), Hefei, China, in 1982, the M.Sc. degree from the North China Institute of Computing Technology, Beijing, China, in 1985, and the Ph.D. degree from USTC, Hefei, in 1990.

He was an Associate Lecturer and Lecturer from 1985 to 1990 with USTC while working on the Ph.D. degree. His Ph.D. work on simulated annealing and evolutionary algorithms was awarded the President's Award for Outstanding Thesis by the Chinese Academy of Sciences. In 1990, he took up a postdoctoral fellowship at the Computer Sciences Laboratory, Australian National University, Canberra, A.C.T., Australia, and continued his work on simulated annealing and evolutionary algorithms. In 1991, he joined the Knowledge-Based Systems Group, Commonwealth Scientific and Industrial Research Organization (CSIRO) Division of Building, Construction, and Engineering, Melbourne, Vic., Australia, working primarily on an industrial project on automatic inspection of sewage pipes. In 1992, he returned to Canberra to take up a lectureship at the School of Computer Science, University College, University of New South Wales, Australian Defence Force Academy, Kensington, N.S.W., Australia, where he was later promoted to Senior Lecturer and Associate Professor. Attracted by the English weather, he moved to the University of Birmingham, Birmingham, U.K., as a Professor (Chair) of computer science on April 1, 1999. He is currently the Director of the Centre of Excellence for Research in Computational Intelligence and Applications, Birmingham, a Distinguished Visiting Professor of the University of Science and Technology of China, Hefei, and a Visiting Professor of three other universities. He has been invited to give more than 45 invited keynote and plenary speeches at conferences and workshops worldwide. He has more than 200 refereed research publications. His research has been supported by research councils, government organizations, and industry (more than £ 4M in the last four years). His major research interests include evolutionary computation, neural network ensembles, and their applications.

Prof. Yao serves as the Editor-in-Chief of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, an associate editor or editorial board member of several other journals, and the Editor of the World Scientific book series on "Advances in Natural Computation." He is also a Distinguished Lecturer of the IEEE Computational Intelligence Society. He is the recipient of the 2001 IEEE Donald G. Fink Prize Paper Award for his work on evolutionary artificial neural networks.