IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 15, 2020, accepted March 6, 2020, date of publication March 11, 2020, date of current version March 23, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2980135

An Improved Simulated Annealing Algorithm With Excessive Length Penalty for Fixed-Outline Floorplanning

ZHIPENG HUANG[1], ZHIFENG LIN[2], ZIRAN ZHU[1], AND JIANLI CHEN[1,3]

[1]Center for Discrete Mathematics and Theoretical Computer Science, Fuzhou University, Fuzhou 350000, China
[2]College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350000, China
[3]State Key Laboratory of ASIC and System, Fudan University, Shanghai 200000, China

Corresponding author: Jianli Chen (jlchen@fzu.edu.cn)

This work was supported in part by the Fujian Science Fund for Distinguished Young Scholars under Grant 2019J06010, and in part by the National Science Foundation of China under Grant 61977017.

ABSTRACT In addition to wirelength and area, modern floorplans need to consider various constraints such as fixed-outline. To handle the fixed-outline floorplanning optimization problem efficiently, we propose an improved simulated annealing (SA) algorithm, which optimizes the area, the total wirelength, and the prescribed outline constraints at the same time. In order to enhance the effectiveness of SA algorithm, we propose a novel feasible solution strategy which ensures that viable solution would be found at all times. Moreover, we propose a new penalty function to better solve the prescribed outline constraint. It consists of a violation area function to prevent modules from moving to the prescribed outline, and an excessive violation function to enable the modules to move close to the optimal positions. Experimental results show that the proposed algorithm is effective and efficient to obtain a fixed-outline floorplan, and achieves a 100% success rate on each benchmark in different aspect ratios.

INDEX TERMS Floorplanning, fixed-outline, simulated annealing algorithm, excessive violation function, feasible solution strategy.

I. INTRODUCTION

In physical design, floorplanning is the first and crucial process to design the floorplan of a chip. Floorplanning determines the topological structure of the floorplan according to area and interconnection estimation of a chip. It provides valuable insight into the hardware decisions and estimation of various costs [1]. It is also important for choosing design alternatives in the early stages which are likely to produce optimal designs [2].

The objective of the conventional floorplanning is to find a legal solution that minimizes certain design metrics, while ensuring no block overlapping with the others. The optimization metrics include area, wirelength, etc. However, with the rapid advancement of technology, many other design constraints and objectives must be taken into consideration to obtain a desirable floorplan. For example, the boundary rectangle of the fixed-outline floorplanning is specified before floorplanning. Moreover, we should determine the dimension of chip and the package before implementing floorplanning, to make it more practical.

For fixed-outline floorplanning, the non-overlapping constraint and fixed outline constraint should be satisfied simultaneously [3]. Boundary rectangle, especially tight boundary, makes floorplanning design more complex, which can not be solved by existing methods. In this paper, we introduce a new penalty function and an improved simulated annealing algorithm to minimize the area and total wirelength of a chip subject to non-overlapping constraint and fixed outline constraint.

A. PREVIOUS WORKS

As the fixed-outline floorplanning problem is becoming more practical, various algorithms have been proposed in the previous works. Hoo *et al.* in [4] proposed a dynamic programming enumeration clustering method to decrease the worse case of time complexity and running time. In [5], the authors presented a novel algorithm, which starts with a sub-example

The associate editor coordinating the review of this manuscript and approving it for publication was Sun-Yuan Hsieh.

of the prescribed floorplan and augments until a candidate solution is obtained.

In addition, the simulated annealing (SA) based algorithms have shown their effectiveness and efficiency for the fixed-outline floorplanning problem. Based on sequence pair [6], the authors in [7] presented a new objective function, and applied a slack-based method to enhance the local search of simulated annealing algorithm. In [8], Roy *et al.* applied min-cut algorithm and simulated annealing algorithm to handle the fixed-outline floorplanning problem. The authors in [9] presented a new perturbation in which the locations of the modules are enumerated to accelerate the search process of simulated annealing algorithm. In [10], the authors proposed a novel fixed-outline floorplanner which applies simulated annealing algorithm to optimize area and wirelength until the utilization ratio of area is 100% as possible. The authors in [11] presented an improved simulated annealing algorithm and an area model to deal with the fixed-outline floorplanning problem. Furthermore, the authors in [12] introduced a modified simulated annealing algorithm and an excessive area model to handle the problem.

Nowadays, SA algorithm has been widely used in very large scale integration (VLSI) design, image recognition, neural network computer research, TSP, knapsack problems and so on. Simulated annealing is a general probability method with random selections, which is used to search the optimal solution of a statement in a large search space. The basic ingredients for SA include solution space, neighborhood structure, cost function, and annealing schedule. The most important part is how to find the feasible solutions in solution space. Since the solution spaces are very large in modern VLSI problems, it is impossible to explore all solutions and even can not find a feasible solution. In order to tackle this problem, for example, based on geometric method, the authors in [22] presented rollback method to deal with the solution space which includes infeasible solutions. Hence, proposing an effective algorithm to address the fixed-outline floorpalnning problem is an enormous challenge.

B. OUR WORK

As the scales of the problem and constraints increase, it's difficult to find feasible solutions. To address the problem effectively, we present a novel feasible solution strategy in SA algorithm. In addition, we introduce a novel penalty function and an improved simulated annealing algorithm to minimize the area and total wirelength of a chip subject to non-overlapping constraint and fixed outline constraint.

Our main contributions can be summarized as follows:

- A novel feasible solution strategy is used to check whether a new neighbor solution is feasible or not. This strategy prevents from transforming a feasible solution into an infeasible one, which enhances the effectiveness of our algorithm. Based on B*-tree representation, our feasible solution strategy based SA algorithm can successfully deal with the fixed-outline floorplanning problem. Compared with other strategies, our strategy can get lower cost.
- The excessive violation function and the area function are integrated in our penalty function. A new excessive violation function is proposed to estimate the length of each module beyond the prescribed outline. The quadratic penalty function model is adopted to penalize the excessive length generated by each module beyond the outline, and enable the modules to move close to the optimal positions. And the area function is used to prevent modules from moving to the prescribed outline. Compared with the leading tools, our function can increase the success rate to 100 percent.
- The comparisons of experiment show the effectiveness and efficiency of the proposed algorithm. For area optimization, the average area values calculated by our algorithm are the minimum among 11 benchmarks. For wirelength optimization, our algorithm could obtain the minimum average wirelength values for all the instances. For co-optimization between area and wirelength, there are 13 cases with the minimum cost for our algorithm.

The remaining structure is assigned as follows. Section II is the preliminaries. Section III presents our method for the fixed-outline floorplanning. Experimental results and conclusions are made in Section IV and Section V, respectively.

II. PRELIMINARIES

A. PROBLEM STATEMENT

Let $M = \{m_1, m_2, \ldots, m_n\}$ be a set of rectangular modules, and the width and height of m_i are w_i and h_i, respectively. $N = \{N_1, N_2, \ldots, N_e\}$ is the netlist that specifies the interconnections between the modules in M. The left bottom coordinate of m_i is denoted as (x_i, y_i). W_i denotes the wirelength of net N_i. For the prescribed outline, its width and height are denoted as W_0 and H_0, respectively. The objective of the fixed-outline floorplanning problem is to find a floorplan F with the width W' and height H' such that the total area and total wirelength of F are minimized, and the following constraints are satisfied.

1) Every module must be placed horizontally or vertically.

2) No module is overlapped with each other.

3) All modules must be placed within the prescribed outline.

In order to judge whether the floorplan is better or not, we first estimate the total area A and total wirelength W of F in this section. The total area A of a floorplan is estimated by the area of the minimum rectangle that encloses the candidate floorplan. Thus, it can be formulated as follows:

$$A = W' \times H', \tag{1}$$

where W' and H' are the width and hight of the smallest enclosing rectangle, respectively. The wirelength of each net N_i is calculated by the half-perimeter

wirelength (HPWL) [14], which is defined as follows:

$$W_i = \max_{m_i,m_j\in N_i} |x_i - x_j| + \max_{m_i,m_j\in N_i} |y_i - y_j|. \quad (2)$$

Thus the total wirelength W of F is calculated by

$$W = \sum_{1\le i\le e} W_i. \quad (3)$$

Next, to describe the non-overlapping constraint specifically, the overlap region of the intervals $[L_1, R_1]$ and $[L_2, R_2]$ can be formulated as follows [15]:

$$\Theta_0([L_1, R_1], [L_2, R_2]) = [\min(R_1, R_2) - \max(L_1, L_2)]^+, \quad (4)$$

where

$$[a]^+ = \begin{cases} a, & if \quad a > 0; \\ 0, & if \quad a \le 0. \end{cases} \quad (5)$$

Thus the overlap area of m_i and m_j is defined by

$$\Theta_{ij}(x_i, y_i, x_j, y_j) = \Theta_0([x_i, x_i + w_i], [x_j, x_j + w_j]) \times \Theta_0([y_i, y_i + h_i], [y_j, y_j + h_j]). \quad (6)$$

Moreover, the prescribed outline constraint can be

$$0 \le x_i, x_i + w_i \le W_0, \ 0 \le y_i, y_i + h_i \le H_0, \quad 1 \le i \le n. \quad (7)$$

Finally, the mathematical formulation of the fixed-outline floorplanning problem is shown as

$$\min \ A + W$$
$$s.t. \begin{cases} \Theta_{ij}(x_i, y_i, x_j, y_j) = 0, & 1 \le i \le n, 1 \le j \le n, i \ne j; \\ 0 \le x_i, x_i + w_i \le W_0, & 1 \le i \le n; \\ 0 \le y_i, y_i + h_i \le H_0, & 1 \le i \le n. \end{cases} \quad (8)$$

For the fixed-outline floorplanning, given aspect ratio $R = H_0/W_0$ and the allowable maximum dead space Π, H_0 and W_0 can be obtained by

$$H_0 = \sqrt{(1+\Pi) \cdot A_{total} \cdot R}, \ W_0 = \sqrt{(1+\Pi) \cdot A_{total}/R}, \quad (9)$$

where A_{total} is the sum of area of all modules in M and

$$A_{total} = \sum_{1\le i\le n} w_i \times h_i. \quad (10)$$

Actually, the fixed-outline floorplanning problem can be regarded as an extension of the strip packing problem in engineering [16]. The similar part between two problems is the constraint that no module overlaps with each other. However, the objective of the fixed-outline floorplanning problem is different from that of the strip packing problem in terms of the following features:

- For fixed-outline floorplanning problem, the constraints also include the outline constraint that all modules must be placed into the prescribed outline.
- In the fixed-outline problem, the overlapping constraint function is non-convex and nonsmooth [17], which is hard to solve directly, especially for large scale circuit.
- In the advanced circuit design, many other design rules are considered in the floorplanning stage, such as connectivity, power consumption and delay [18]–[20].

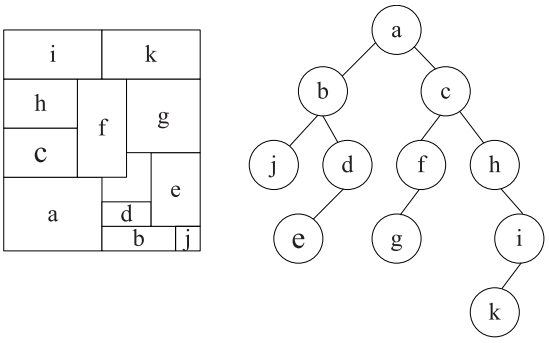


FIGURE 1. A feasible floorplan and its corresponding *B*-tree*.

B. B*-TREE

For a feasible fixed-outline floorplan, we can construct a unique *B*-tree*, and vice versa [13]. Actually, the construction between a *B*-tree* and its floorplan is similar to the depth first search (DFS). The detailed procedure for the transformation between a floorplan and its *B*-tree* is described in [13]. A fixed-outline floorplan is feasible if the outline constraint is satisfied and any module could neither be removed left nor be removed down. Figure 1 shows a feasible fixed-outline floorplan and its *B*-tree*.

Based on the characteristics of the ordered binary tree, the computation complexity of search, insertion and deletion operations in a *B*-tree* is $O(1)$, $O(1)$ and $O(n)$, respectively. Moreover, a *B*-tree* and its feasible floorplan have a one-to-one correspondence relationship, which results in that the size of solution space of *B*-tree* is smaller than that of other representations. The detailed comparison of various representations can be found in [13]. Among these methods, *B*-tree* is considered as one of the most effective representations in describing geometry relationship among modules.

In order to produce a neighbor solution, the authors in [13] presented following three operation modes to disturb a *B*-tree* to another one: rotate a module, remove a module to another place, and swap two modules.

III. OUR ALGORITHM

In this paper, the overall flow of our algorithm is shown in Figure 2. It starts with a set of rectangular modules, corresponding circuit netlist and prescribed outline. And we also initialize some parameters including initial temperature, termination of the temperature, rate of convergence and rejecting inferior solutions. After that, a complete binary tree is taken as an initial solution. To complete the above processes, our algorithm consists of four main parts: (1) novel feasible solution strategy, (2) cost function estimation, (3) the cooling schedule, and (4) the general improved simulated annealing algorithm. We will detail these four major parts in the following subsections.

A. NOVEL FEASIBLE SOLUTION STRATEGY

In this section, we present a novel feasible solution strategy for simulated annealing algorithm. The detailed descriptions are shown as follows.

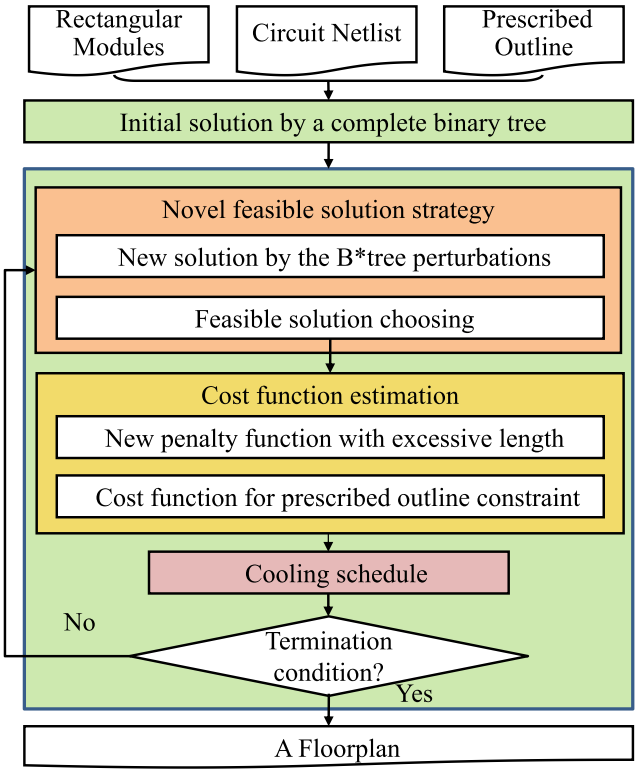


FIGURE 2. Our algorithm flow.

TABLE 1. Comparison with the three strategies.

Strategy	Character of the solution	Obtained feasible solution	Timecost
Geometric strategy	Last solution	Not always	Normal
Rollback strategy	Last feasible solution	Always but uncertain	Normal
Our feasible solution strategy	First or second feasible solution	100%	Faster

In SA algorithm, a series of neighbor solutions can be generated by the perturbations. However, a neighbor solution may be infeasible since the operation is selected at random. Searching for optimal solution or near optimal solutions would be challenging if the solution space is filled with a large number of infeasible solutions. Therefore, some strategies [22] are presented to evaluate the feasibility of a new solution. What's more, we present a novel feasible solution strategy which is described in Algorithm 1. In this algorithm, a complete binary tree [23] with specified modules is considered as the initial input solution S. Here j is the iteration number, and the specified number t and the probability r in Algorithm 1 are user-defined.

Table 1 compares the differences among our strategy and two common strategies mentioned in [22]. By applying a series of perturbations, the result of geometric strategy is a solution which is the last solution. And the result of rollback strategy is a solution which is the last feasible solution. At last, the result of our novel feasible solution strategy is a solution which is the first or second feasible solution. By analyzing the performances of three strategies, if the solution spaces include feasible solutions, our approach has the following two advantages. On the one hand, if the solution produced by perturbations is infeasible and the prescribed number r is bigger than the random number, geometric strategy and rollback strategy are inefficient to find a feasible solution. However, our feasible solution strategy could ensure that viable solutions would be found at all times. On the other hand, by using the same parameters, if rollback strategy has a feasible solution, our strategy takes less runtime than the other two strategies, which makes it more efficient.

Algorithm 1 Our Feasible Solution Strategy

Input:
An initial input solution S;
Output:
A feasible solution S_j;

```
j ← 0;
while 1 do
  j ← j + 1;
  generate a new solution S_j by the perturbations;
  if S_j is feasible then
    while j ≤ t do
      j ← j + 1;
      generate a new solution S_j by the perturbations;
      if r ≥ (random number) or S_j is feasible then
        break;
      end if
    end while
    break;
  end if
end while
```

B. COST FUNCTION ESTIMATION

In order to better estimate width or height of each module beyond the prescribed outline, we propose a new penalty function with excessive violation function. Based on the excessive violation function, it is easy for our algorithm to find a desirable solution.

1) NEW PENALTY FUNCTION

In the proposed algorithm, we adopt penalization to prevent the modules from moving out of the prescribed outline. Since the penalization is one of the criterions to judge whether a solution is feasible or not, its definition is of great significance. Recently, several function models have been presented for penalization. For example, in [11], the excessive area function is calculated by the difference between the area function of the smallest rectangle which encloses the candidate floorplan and the prescribed outline, which is defined as follows:

$$A' = \begin{cases} 0, & \text{if } H' \le H_0,\ W' \le W_0; \\ (W' - W_0) \cdot H_0, & \text{if } H' \le H_0,\ W' \ge W_0; \\ (H' - H_0) \cdot W_0, & \text{if } W' \le W_0,\ H' \ge H_0; \\ W' \cdot H' - W_0 \cdot H_0, & \text{otherwise.} \end{cases} \quad (11)$$

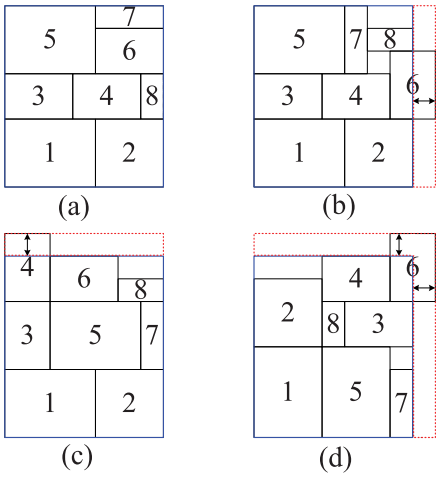


FIGURE 3. Four violation cases. (a) No violation. (b) The width of the 6th module violation. (c) The height of the 4th module violation. (d) Width and height of the 6th module violation.

Based on the area function model [11], we propose a new excessive violation function which emphasizes on the length of each module outside the prescribed outline. The excessive violation function L' is defined as follows:

$$L' = \sum_{i \in S} L_i, \tag{12}$$

where S is the set of modules which exceed the fixed outline, and L_i is the violation function of the *i-th* module outside the prescribed outline. According to the coordinate of module i, L_i can be defined in (13) and the corresponding cases are depicted in Figure 3.

$$L_i = \begin{cases} 0, \\ \quad \text{if } y_i + h_i \le H_0, x_i + w_i \le W_0; \\ (x_i + w_i - W_0)^2, \\ \quad \text{if } y_i + h_i \le H_0, x_i + w_i \ge W_0; \\ (y_i + h_i - H_0)^2, \\ \quad \text{if } y_i + h_i \ge H_0, x_i + w_i \le W_0; \\ (x_i + w_i - W_0)^2 + (y_i + h_i - H_0)^2, \\ \quad \text{otherwise.} \end{cases} \tag{13}$$

In Figure 3, the rectangular whose boundary is drawn in blue represents the prescribed outline. The rectangular with the red dotted line boundary denotes the area function A' which is defined in (11). Additionally, the modules with the arrow denote that they exceed the prescribed outline. Figure 3(a) shows that all modules are able to be placed into the prescribed outline. From Figure 3(b), it can be seen that the width of module 6 exceeds the prescribed outline. Similarly, for the module 4 in Figure 3(c), its height exceeds the prescribed outline. However, though there is only one module outside the prescribed outline in Figure 3(d), both the width and height of the module 6 exceed the prescribed outline.

When only the area model [11] is considered, we note that the model cannot exactly deal with the prescribed outline constraint from Figure 4. In Figure 4(a) and 4(b), the number of modules beyond the outline are one and two, respectively. If we look at the two cases, the area values are identical when only area model is taken into account. However, the floorplan of Figure 4(b) is distinctly worse than that of Figure 4(a). Therefore, it is necessary to develop a more accurate penalization which takes the excessive length of each module into consideration.

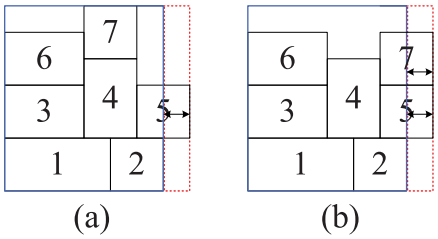


FIGURE 4. The different violation cases. (a) One module exceeding the outline. (b) Two modules beyond the outline. Moreover, for (a) and (b), there is the identical excessive area value when only area model is taken into account.

According to the above analysis, our proposed excessive penalization is defined as follows:

$$\Phi = A' + L'. \tag{14}$$

In this penalization, the area function and the excessive length function are integrated. The area function is used to prevent modules from moving to the prescribed outline, which gives the global search direction. What's more, the quadratic penalty function model is adopted to deal with the excessive length generated by each module beyond the outline in (13). This function can provide the local search direction and enable the modules moving close to the optimal positions.

2) COST FUNCTION

For the fixed-outline floorplanning problem, the objective is to optimize the total area and total wirelength of a chip such that the constraints mentioned in Section II-A are satisfied. The area A of a floorplan is estimated by the area of the minimum rectangle which encloses the candidate floorplan, and the total wirelength of a chip is calculated by the sum of W_i $(1 \le i \le e)$. Besides, a penalty function Φ is used to describe the prescribed outline constraint, which is shown in Section III-B.1. Hence the cost function is defined as follows:

$$cost(F) = \alpha \times \frac{A}{A^*} + \beta \times \frac{W}{W^*} + \gamma \times \frac{\Phi}{\Phi^*}. \tag{15}$$

where α, β and γ are weights allocated to the area, wirelength and penalty function, respectively. In addition, these coefficients satisfy the relationship that $0 \le \alpha, \beta \le 1, \alpha + \beta = 1$, where γ is a user-defined parameter. The area A, wirelength W and penalization Φ are separately normalized with their minimum value A^*, W^* and Φ^*.

C. THE COOLING SCHEDULE

Simulated annealing algorithm inspired by the physical phenomena in fluids solidification process [24] is considered as

an efficient and general probabilistic algorithm for handling the floorplanning problem. Different from greedy algorithm, SA has a probability to receive a poor solution. This probability is related to the temperature T and the difference Δ between the last solution and the current solution, where Δ is calculated by the following equation:

$$\Delta = cost_{current} - cost_{last}. \tag{16}$$

Classical simulated annealing algorithm estimate the temperature T by the following equation:

$$T = \theta \times T, \tag{17}$$

where θ is the coefficient between 0 and 1. From this formula, it can be seen that the temperature reduces at first. It makes that the probability approaches to 0, which results in accepting few poor solutions. In this way, the classical simulated annealing algorithm may fail to find a desirable solution.

To improve simulated annealing algorithm, an effective hybrid simulated annealing algorithm [24] is introduced to address non-slicing VLSI floorplanning problem. Experiment results show that the hybrid simulated annealing algorithm can produce optimal or nearly optimal chip areas for all the tested benchmarks [21]. On the basis of the fast simulated annealing algorithm [22] and hybrid simulated annealing algorithm [24], we propose an improved simulated annealing algorithm to find a high quality solution. Similar to the annealing measure [22], the cooling schedule in our improved simulated annealing algorithm consists of three parts:

1) The stochastic search process. The process is identical to the classical simulated annealing algorithm. In this part, the temperature is high, which makes the acceptance probability for poor solution also high (Close to 1).

2) The local search process. In this process, the temperature is controlled by the parameter λ which approaches 0. Thus few poor solutions can be accepted according to the lower acceptance probability.

3) The uphill search process. In this process, with the growing temperature, the probability of accepting inferior solutions is also increasing. It makes many inferior solutions accepted. In other words, the uphill search process contributes to preventing from falling into the local minimal solution.

The calculation of cooling temperature is shown in Algorithm 2.

Algorithm 2 The Cooling Schedule

Require: The parameter c and the parameter P;
Ensure: T_n, T_{actual};
1: $T_n = avg/(log(P))$;
2: **if** $n = c$ **then**
3: $T_n = T_n \times P^m \times avg_{cost}/(\lambda \times n)$;
4: $T_{actual} = estimate_{avg}/T_n$;
5: **end if**
6: **if** $n > c$ **then**
7: $T_n = T_n \times avg_{cost}/n$;
8: $T_{actual} = estimate_{avg}/T_n$;
9: **end if**

Algorithm 3 Outline of Our Algorithm

Require: The initial solution I_0 and its corresponding *B*-tree* J_0;
Ensure: The best solution I_1;
1: the operation number n on *B*-tree* is set to 0 at the beginning;
2: the uphill number of accepting poor solution and the number of rejecting poor solution are initialized to 0;
3: empty list L;
4: **while** ($reject_{rate} < conv_{rate}$ and $T_{actual} > T_{term}$) or $n >= 15$ **do**
5: $n \leftarrow n + 1$;
6: produce a new solution I', and its corresponding to *B*-tree* J;
7: call Algorithm 1, and return a feasible solution I';
8: calculate the cost value of I' and put I' in L;
9: $\Delta \leftarrow cost(I') - cost(I_0)$;
10: **if** $\Delta \leq 0$ **then**
11: $I_0 \leftarrow I', J_0 \leftarrow J, I_1 \leftarrow I'$;
12: **else**
13: randomly select a real μ between 0 and 1;
14: **if** $\mu < e^{-\frac{\Delta}{T}}$ **then**
15: $I_0 \leftarrow I', J_0 \leftarrow J$;
16: **else**
17: $reject \leftarrow reject + 1$;
18: **end if**
19: **end if**
20: update T_n by Algorithm 4;
21: $reject_{rate} \leftarrow reject/n$;
22: **end while**
23: return I_1.

In Algorithm 2, n is the iteration index of simulated annealing algorithm. The average uphill cost and the initial acceptance probability for inferior solutions are denoted as avg and P, respectively. avg_{cost} defined in (16) is the average difference in the current temperature. λ and c are user-defined parameters. In the annealing process, λ is the coefficient which decrease the temperature rapidly to 0. In our algorithm, the parameter λ is set to 100. In addition, $estimate_{avg}$ is a constant to control the temperature, and T_{actual} is the actual temperature.

D. IMPROVED SIMULATED ANNEALING ALGORITHM

Based on *B*-tree* representation, a new penalty function and an improved simulated annealing are presented to handle the fixed-outline floorplanning. First, a complete binary tree is taken as an initial solution in our improved algorithm. Second, a novel feasible solution strategy is introduced since a neighboring solution produced by the perturbation on *B*-tree* may be infeasible. Third, the cooling schedule is presented, which makes our algorithm more efficient. Based on these modified methods, our improved simulated annealing algorithm is applied to find an optimal solution. Algorithm 3 shows the detailed description of our algorithm.

In Algorithm 3, $conv_{rate}$ is the rate of convergence, and $reject_{rate}$ is the rate of rejecting inferior solutions, $T_{term} = 0.1$ is the termination of the temperature. This makes the solution be enough cooling.

IV. EXPERIMENTAL RESULTS

In this section, we evaluate the performance of our algorithm on the GSRC benchmarks [25]. The proposed algorithm is implemented in C++ programming language and performed on an Intel Core (TM) i5-4210M CPU @2.6 GHz CPU and 4GB RAM. Besides, the Dead Space Π of all benchmarks are arranged to 10%. The aspect ratio R of each benchmark is respectively arranged to 1.0, 2.0 and 3.0.

A. EFFECTIVENESS OF THE IMPROVED SIMULATED ANNEALING ALGORITHM WITH OUR NOVEL FEASIBLE SOLUTION STRATEGY

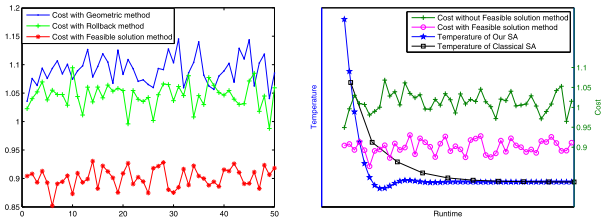
In order to validate the effectiveness of our feasible solution strategy, we also implement geometric strategy and rollback strategy for comparison purpose. Among these three strategies, a complete binary tree with the specified modules is considered as the initial input solution S. The parameter t of three strategies is set to 30 and r is set to 0.15.

The cost values which discussed in (15) are also taken into consideration. We run each strategy for 50 times on benchmark n30. The cost curve shown in Figure 5(a) demonstrates the computation time of different strategies. The blue, green and red curves represent the cost change of geometric strategy, rollback strategy, and our feasible strategy, respectively. From Figure 5(a), it can be seen that the average cost value calculated by our feasible solution strategy is better.

In addition, compared with the classical simulated annealing algorithm, our feasible solution strategy based algorithm has high probability to find a high quality solution, which is illustrated in Figure 5(b). Figure 5(b) depicts the relationship between temperature and cost which is calculated by the area and wirelength. In this figure, the curve drawn in black denotes the temperature change of the classical simulated annealing algorithm. The blue curve denotes the temperature change of our simulated annealing algorithm. According to the left temperature axis, the metrics of our improved simulated annealing algorithm can be obtained. For example, the temperature drops sharply such that redundant solutions are ignored. The curve drawn in green in Figure 5(b) denotes the cost change without feasible solution strategy. The pink curve denotes the cost change with feasible solution strategy. Similarly, according to the right cost axis, it can be found that the average cost with feasible solution strategy is better.

B. THE VALIDITY OF OUR PENALTY FUNCTION

In this part, we take the penalty function in (14) into account. For convenience, our penalization is denoted by EAL. To evaluate the effectiveness of EAL, we combine classical simulated annealing (SA) with different penalty functions: EL1, EL2, EL3, EL4, EA1 and EA2, which are respectively introduced in [5], [7], [9], [11], [12]. We run each instance 20 times and the experimental results are listed in Table 2. For each benchmark, the maximum runtime is set as 10s, 10s, 30s, 40s and 60s, respectively.



(a) (b)

FIGURE 5. (a) Comparisons of the cost values for n30 when $R = 1$. (b) The temperature and cost comparisons of n30 when $R = 1$.

In Table 2, the success rate (SR) and average runtime (AR) of our penalization are compared with those of other penalty functions. If SR is 100%, it means that the modules for every benchmark are placed into the prescribed outline. The term SA-EAL denotes that the classical simulated annealing is integrated with EAL function model. Similarly, other six items have the same meanings. From Table 2, we can see that for all benchmarks, the success rates of SA-EA2 and SA-EAL are 100%. In other words, for all instances, SA-EA2 and SA-EAL have a fair chance of finding a high quality solution. Furthermore, compared with SA-EA2, SA-EAL takes less runtime on the whole. For instance, the runtime on n30 is less than that of SA-EA2 when aspect ratio is 1.0 and 3.0, respectively. For n50, although the runtime is larger than that of SA-EA2 when aspect ratio is 3.0, the runtime is the least when aspect ratio is respectively equal to 1.0 and 2.0. Actually, it only takes more 0.02s than SA-EA2 when aspect ratio is 3.0. For n100, the runtime on n30 is the least for all aspect ratios. For n200, it costs more 0.53s than SA-EA2 when aspect ratio is 1.0. But the runtime is less when aspect ratio is respectively 2.0 and 3.0. For n300, when aspect ratio is 1.0, SA-EA2 saves more 0.66s than that of our proposed function. However, in other cases, the benchmark n300 has the similar situation as the benchmark n200. From Table 2, it could be inferred that our penalization could more efficiently cope with the fixed outline floorplanning problem.

C. AREA OPTIMIZATION

In this part, we lay particular emphasis on the area and fixed outline constraint. We make a comparison between our algorithm and MSA [12]. The results are listed in Table 3.

In Table 3, AA is the average area value. Note that the success rates calculated by the algorithms with our penalization are 100%, while those estimated by other function models are not completely 100%. Actually, as the number of the modules increases, the success rates are worse. For instance, the success rates on n30, n50 and n100 calculated by the algorithm with EA1 function are 100%. However, the success rates on n200 and n300 calculated by SA-EA1 and MSA-EA1 are not completely 100%.

TABLE 2. Comparison results of the success rate and average runtime on the GRSN benchmarks.

Circuit	R	SA-EL1		SA-EL2		SA-EL3		SA-EL4		SA-EA1		SA-EA2		SA-EAL	
		SR(%)	AR(s)	SR(%)	AR(s)	SR(%)	AR(s)	SR(%)	AR(s)	SR(%)	AR(s)	SR(%)	AR(s)	SR(%)	AR(s)
n30	1.0	100	0.24	100	0.23	100	0.26	100	0.22	100	0.26	100	0.23	100	**0.10**
	2.0	100	0.38	100	0.21	100	0.31	100	0.27	100	0.25	100	**0.15**	100	0.17
	3.0	100	0.30	100	0.29	100	0.32	100	0.25	100	0.26	100	0.26	100	**0.20**
n50	1.0	100	0.64	100	0.56	100	0.78	100	0.51	100	0.46	100	0.35	100	**0.22**
	2.0	100	0.91	100	0.78	100	0.69	100	0.61	100	0.70	100	0.44	100	**0.21**
	3.0	100	0.88	100	0.74	100	0.92	100	0.79	100	0.81	100	**0.37**	100	0.39
n100	1.0	100	2.51	100	6.01	100	2.59	100	3.00	100	1.89	100	1.38	100	**1.23**
	2.0	100	5.80	100	6.75	100	4.89	100	4.28	100	2.05	100	1.91	100	**1.38**
	3.0	100	8.99	100	7.26	100	7.82	100	6.21	100	7.11	100	2.23	100	**1.35**
n200	1.0	95	12.83	30	37.76	100	9.87	95	18.27	100	9.62	100	**4.22**	100	4.75
	2.0	30	32.12	10	39.19	95	13.42	60	31.53	90	13.08	100	5.87	100	**4.11**
	3.0	0	40.00	0	40.00	35	31.29	5	39.79	25	31.95	100	6.76	100	**5.19**
n300	1.0	100	33.35	0	60	90	33.55	75	38.84	100	29.98	100	**11.11**	100	11.77
	2.0	25	53.44	0	60	85	28.58	30	50.24	75	31.28	100	17.58	100	**11.11**
	3.0	5	57.61	0	60	55	35.76	5	57.92	40	45.72	100	13.99	100	**10.9**

TABLE 3. Comparison results of the average area on the GRSN benchmarks.

Circuit	R	SA-EA1		SA-EA2		SA-EAL		MSA-EA1		MSA-EA2		MSA-EAL		Our SA-EA1		Our SA-EA2		Our SA-EAL	
		AA	SR	AA	SR	AA	SR	AA	SR	AA	SR	AA	SR	AA	SR	AA	SR	AA	SR
n30	1.0	217723	100	218075	100	224172	100	218425	100	217778	100	224651	100	220942	100	**217120**	100	217592	100
	2.0	217263	100	218669	100	221409	100	216859	100	217793	100	220038	100	219854	100	217365	100	**216464**	100
	3.0	218292	100	217357	100	223560	100	216544	100	217244	100	223284	100	220311	100	217008	100	**216529**	100
n50	1.0	205669	100	205947	100	214360	100	205871	100	205472	100	214353	100	207708	100	205660	100	**204295**	100
	2.0	205652	100	206396	100	213840	100	205836	100	205493	100	214837	100	209538	100	206032	100	**205328**	100
	3.0	205379	100	206599	100	213586	100	205469	100	205583	100	213855	100	207876	100	205556	100	**205352**	100
n100	1.0	**186761**	100	187124	100	194916	100	187789	100	187430	100	194028	100	190207	100	188320	100	187880	100
	2.0	186883	100	186728	100	194680	100	186829	100	**186518**	100	194680	100	189616	100	187200	100	187200	100
	3.0	186198	100	186800	100	195072	100	186818	100	**185990**	100	195072	100	191550	100	187170	100	186624	100
n200	1.0	187802	95	186756	100	191840	100	199094	65	187261	100	191400	100	187295	100	187085	100	**185736**	100
	2.0	188710	75	185834	100	191270	100	194161	20	185962	100	191580	100	186942	100	186067	100	**184450**	100
	3.0	195446	15	185247	100	190762	100	197006	0	185200	100	191015	100	186001	100	186136	100	**183931**	100
n300	1.0	294671	100	295333	100	298660	100	303404	25	294264	100	298660	100	294022	100	292968	100	**290485**	100
	2.0	292039	95	293261	100	299538	100	308033	15	292323	100	298764	100	290729	100	291412	100	**288702**	100
	3.0	292080	95	291755	100	299568	100	315086	0	291557	100	299252	100	291259	100	290880	100	**287244**	100

Additionally, from Table 3, it can be found that the success rates of the algorithms with EA2 are also 100%. However, compared with SA-EA2, for most benchmarks the average area values of MSA are worse than those of our algorithm, which indicates that our algorithm is better than MSA. In other words, paralleled with other algorithms, the average area values calculated by our algorithm are the minimum on 11 benchmarks. From Table 3, it can be seen that our algorithm has a fair chance of getting desirable solutions.

D. WIRELENGTH OPTIMIZATION

In this part, the wirelength and fixed outline constraint are emphasized. We compare our algorithm with MSA [12]. The results are displayed in Table 4.

In Table 4, AWL is the average wirelength value. It should be noted that the success rates of the algorithms with our penalization are 100%. Additionally, the success rates of the algorithms with EA2 function are also 100%. However, the success rates of the algorithms with EA1 function are not completely equal to 100%. Actually, the success ratio of the algorithms with EA1 function decreases as the increasing number of the modules. Since the objective of fixed-outline floorplanning problem is finding a desired floorplan that all modules are placed into the prescribed outline, EA2 function and our penalization are better than EA1 function.

Additionally, though the success rates got by MSA are equal to 100%, the average wirelength values for all benchmarks are worse than those obtained by our algorithm. In other words, our algorithm could obtain the minimum average wirelength values for all the instances. This indicates that our algorithm is more efficient to deal with the wirelength optimization problem under the fixed outline constraint.

E. CONCURRENT AREA AND WIRELENGTH OPTIMIZATION

In this part, the area, total wirelength and fixed outline constraint are considered at the same time. We compare our algorithm with MSA [12]. The results are reported in Table 5.

In Table 5, AOFV is the average cost value. From Table 5, it can be seen that the success ratios of the algorithms with EAL function are 100%. Besides, the success ratios of the algorithms with EA2 function are also 100%. But the success ratio of the algorithms with EA1 function is not completely 100%. Actually, as the number of the modules increases, the success ratio of EA1 drops. These illustrate that EA1 function is worse than EA2 function.

Though the success rates of MSA are 100%, the average cost values of our algorithm are better than those of MSA on the whole. In fact, for our algorithm, there are 13 cases with the minimum cost. In comparison with MSA, when R is 1.0, the cost value of our algorithm on n200 is worse.

TABLE 4. Comparison results of the average wirelength on the GRSN benchmarks.

Circuit	R	SA-EA1		SA-EA2		SA-EAL		MSA-EA1		MSA-EA2		MSA-EAL		Our SA-EA1		Our SA-EA2		Our SA-EAL	
		AWL	SR	AWL	SR	AWL	SR	AWL	SR	AWL	SR	AWL	SR	AWL	SR	AWL	SR	AWL	SR
n30	1.0	120623	100	119436	100	115530	100	120119	100	121023	100	118385	100	119402	100	119402	100	**115111**	100
	2.0	123951	100	125204	100	120677	100	125098	100	124655	100	119584	100	125393	100	124548	100	**116585**	100
	3.0	133555	100	133135	100	130619	100	134339	100	132013	100	130593	100	131808	100	135316	100	**123715**	100
n50	1.0	155227	100	153848	100	154133	100	155368	100	154204	100	152134	100	152894	100	156066	100	**147249**	100
	2.0	158987	100	164483	100	157407	100	157518	100	156947	100	158994	100	160789	100	162782	100	**150668**	100
	3.0	169358	100	170322	100	180696	100	168844	100	168222	100	177109	100	172382	100	174051	100	**161605**	100
n100	1.0	241977	100	243782	100	234017	100	239211	100	241323	100	251687	100	240946	100	245880	100	**236875**	100
	2.0	248516	100	257461	100	271057	100	247809	100	247150	100	265741	100	258880	100	269545	100	**238797**	100
	3.0	269278	100	266581	100	284349	100	260735	100	266090	100	291348	100	305821	100	279266	100	**246404**	100
n200	1.0	463923	95	458269	100	472041	100	529162	45	450291	100	495170	100	450941	100	456738	100	**433367**	100
	2.0	474813	90	465320	100	510356	100	591249	0	467029	100	481488	100	468728	100	467049	100	**455293**	100
	3.0	545192	65	505975	100	505429	100	635068	0	506574	100	540473	100	499649	100	500965	100	**474661**	100
n300	1.0	680069	100	641884	100	740899	100	765977	45	639793	100	709613	100	639518	100	633150	100	**613523**	100
	2.0	687059	90	673267	100	758453	100	872540	10	684506	100	768592	100	667876	100	671823	100	**646088**	100
	3.0	821299	60	737587	100	835486	100	956978	0	731336	100	777280	100	719196	100	721808	100	**677303**	100

TABLE 5. Comparison results of the average cost on the GRSN benchmarks.

Circuit	R	SA-EA1		SA-EA2		SA-EAL		MSA-EA1		MSA-EA2		MSA-EAL		Our SA-EA1		Our SA-EA2		Our SA-EAL	
		AOFV	SR	AOFV	SR	AOFV	SR	AOFV	SR	AOFV	SR	AOFV	SR	AOFV	SR	AOFV	SR	AOFV	SR
n30	1.0	1.058293	100	1.047826	100	1.141661	100	1.046095	100	1.045172	100	0.965485	100	0.955161	100	1.039902	100	**0.837276**	100
	2.0	1.076343	100	1.084704	100	0.997892	100	1.077146	100	1.080114	100	1.001022	100	1.110268	100	1.077985	100	**0.960701**	100
	3.0	1.118898	100	1.125955	100	1.074910	100	1.123375	100	1.115012	100	1.298842	100	1.140878	100	1.096033	100	**1.047455**	100
n50	1.0	0.904454	100	0.899078	100	0.797563	100	0.894795	100	0.900166	100	0.879315	100	0.914046	100	0.818892	100	**0.761110**	100
	2.0	0.910948	100	0.923015	100	0.972555	100	0.909410	100	0.913261	100	0.891023	100	0.817864	100	0.924986	100	**0.762671**	100
	3.0	0.956104	100	0.954699	100	0.948996	100	0.947424	100	0.940996	100	0.972609	100	0.905589	100	0.924743	100	**0.847355**	100
n100	1.0	0.687007	100	0.689286	100	0.659825	100	0.682224	100	0.685029	100	0.666316	100	0.672132	100	0.676334	100	**0.581281**	100
	2.0	0.702461	100	0.703290	100	0.651336	100	0.700334	100	0.697567	100	0.742848	100	0.686007	100	0.782674	100	**0.627419**	100
	3.0	0.729840	100	0.725235	100	0.682420	100	0.729039	100	0.720136	100	0.736568	100	0.806380	100	0.722825	100	**0.589543**	100
n200	1.0	0.588953	100	0.591166	100	0.653470	100	0.631980	60	0.590825	100	0.659457	100	**0.525993**	100	0.590600	100	0.563044	100
	2.0	0.608225	90	0.603381	100	0.578871	100	0.694236	0	0.610766	100	0.554795	100	0.618968	100	0.618658	100	**0.561894**	100
	3.0	0.627114	95	0.620991	100	0.654846	100	0.727570	0	0.619072	100	0.670721	100	0.661531	100	0.654697	100	**0.558426**	100
n300	1.0	0.500331	100	0.495205	100	0.572063	100	0.550421	35	0.502643	100	0.562949	100	0.478825	100	0.509442	100	**0.474126**	100
	2.0	0.507663	100	0.510876	100	0.557292	100	0.596934	0	0.511838	100	0.554627	100	0.555886	100	0.479547	100	**0.449449**	100
	3.0	0.534674	95	0.533615	100	0.561319	100	0.614755	10	0.529251	100	0.571350	100	**0.505602**	100	0.512549	100	0.508319	100

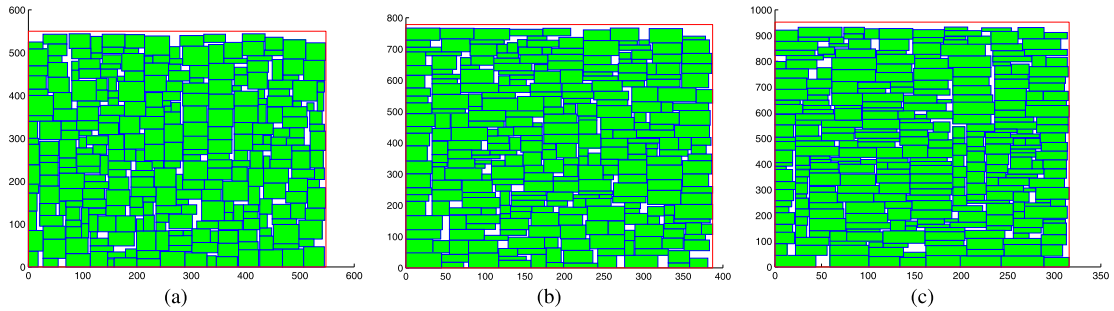


FIGURE 6. A floorplan on n300 with (a) R = 1.0, (b) R = 2.0, and (c) R = 3.0.

Besides, when R is 3.0, the cost values of our algorithm on n300 are inferior. However, in other cases, the cost values are better. These imply that our algorithm is more efficient to seek for desirable solutions. For the purpose of understanding intuitively, the floorplans of n300 with R = 1.0, 2.0 and 3.0 are shown in Figure 6(a)-Figure 6(c), respectively.

V. CONCLUSION

In this paper, we present a new penalty function, and apply an improved simulated annealing algorithm to handle instances. The penalty function is composed of the violation area function and the length function. To calculate the violation area function, all modules are taken as a whole, and used to estimate the area outside the prescribed outline. The length function is employed to estimate the length of each module beyond the prescribed outline. In the proposed algorithm, a feasible solution strategy is used to enhance the effectiveness of our algorithm. This strategy can prevent from transforming a feasible solution into a infeasible solution. Experimental results show that our algorithm is more effective for solving the fixed-outline floorplanning problem compared with other algorithms.

ACKNOWLEDGMENT

The authors would like to thank the editor and anonymous reviewers, whose insightful comments and suggestions helped improving the quality of the manuscript.

REFERENCES

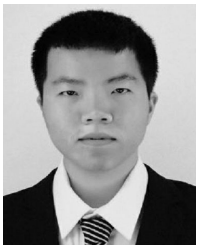
[1] T.-C. Chen and Y.-W. Chang, "Floorplanning," in *Electronic Design Automation: Synthesis, Verification, and Test*. San Francisco, CA, USA: Morgan Kaufmann, 2009, ch. 10, pp. 575–634.
[2] C. J. Alpert and D. P. Mehta, *Handbook of Algorithms for Physical Design Automation*. New York, NY, USA: Auerbach, 2008, pp. 139–257.
[3] S. N. Adya and I. L. Markov, "Fixed-outline floorplanning through better local search," in *Proc. IEEE Int. Conf. Comput. Design, VLSI Comput. Processors. (ICCD)*, Sep. 2001, pp. 328–334.
[4] C.-S. Hoo, J. Kanesan, and H. Ramiah, "Enumeration technique in very large-scale integration fixed-outline floorplanning," *IET Circuits, Devices Syst.*, vol. 8, no. 1, pp. 47–57, Jan. 2014.
[5] R. Liu, S. Dong, X. Hong, and Y. Kajitani, "Fixed-outline floorplanning with constraints through instance augmentation," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 2, 2005, pp. 1883–1886.
[6] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "VLSI module placement based on rectangle-packing by the sequence-pair," *IEEE Trans. Comput.-Aided Design Integr.*, vol. 15, no. 12, pp. 1518–1524, Dec. 1996.
[7] S. N. Adya and I. L. Markov, "Fixed-outline floorplanning: Enabling hierarchical design," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 6, pp. 1120–1135, Dec. 2003.
[8] J. A. Roy, S. N. Adya, D. A. Papa, and I. L. Markov, "Min-cut floorplacement," *IEEE Trans. Comput.-Aided Design Integr.*, vol. 25, no. 7, pp. 1313–1326, Jul. 2006.
[9] S. Chen and T. Yoshimura, "Fixed-outline floorplanning: Block-position enumeration and a new method for calculating area costs," *IEEE Trans. Comput.-Aided Design Integr.*, vol. 27, no. 5, pp. 858–871, May 2008.
[10] O. He, S. Dong, J. Bian, S. Goto, and C.-K. Cheng, "A novel fixed-outline floorplanner with zero deadspace for hierarchical design," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, Nov. 2008, pp. 16–23.
[11] D.-X. Zou, G.-S. Hao, G. Pan, and G.-G. Wang, "An improved simulated annealing algorithm and area model for the fixed-outline floorplanning with hard modules," in *Proc. 3rd Int. Symp. Comput. Bus. Intell. (ISCBI)*, Dec. 2015, pp. 21–25.
[12] D.-X. Zou, G.-G. Wang, G. Pan, and H.-W. Qi, "A modified simulated annealing algorithm and an excessive area model for floorplanning using fixed-outline constraints," *Frontiers Inf. Technol. Electron. Eng.*, vol. 17, no. 11, pp. 1228–1244, Nov. 2016.
[13] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, and S.-W. Wu, "B*-tree: A new representation for non-slicing floorplans," in *Proc. ACM/IEEE Design Autom. Conf.*, Jun. 2000, pp. 458–463.
[14] P.-N. Guo, C.-K. Cheng, and T. Yoshimura, "An O-tree representation of non-slicing floorplan and its applications," in *Proc. ACM/IEEE Design Autom. Conf.*, Jun. 1999, pp. 268–273.
[15] W. Huang, D. Chen, and R. Xu, "A new heuristic algorithm for rectangle packing," *Comput. Oper. Res.*, vol. 34, no. 11, pp. 3270–3280, Nov. 2007.
[16] C. Chu, "Placement," *Electronic Design Automation: Synthesis, Verification, and Test*. San Francisco, CA, USA: Morgan Kaufmann, 2009, ch. 11, pp. 635–686.
[17] Y.-W. Chang, Z.-W. Jiang, and T.-C. Chen, "Essential issues in analytical placement algorithms," *IPSJ Trans. Syst. LSI Design Methodol.*, vol. 2, pp. 145–166, Aug. 2009.
[18] *ISPD 2011 Routability-Driven Placement*. [Online]. Available: http://www.ispd.cc/contests/11/ispd2011_contest.html
[19] *DAC 2012 Routability-Driven Placement Contest*. [Online]. Available: http://archive.sigda.org/dac2012/contest/dac2012_contest.html
[20] *ICCAD 2014 Incremental Timing-Driven Placement*. [Online]. Available: http://cad_contest.ee.ncu.edu.tw/CAD-Contest-at-ICCAD2014/
[21] J. Chen, W. Zhu, and M. M. Ali, "A hybrid simulated annealing algorithm for nonslicing VLSI floorplanning," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 41, no. 4, pp. 544–553, Jul. 2011.
[22] H. Nakano and K. Fujiyoshi, "Improved method of simulated annealing for unreachable solution space," in *Proc. 20th Workshop Synth. Syst. Integr. Mixed Inf. Technol. (SASIMI)*, 2016, pp. 346–351.
[23] T.-C. Chen and Y.-W. Chang, "Modern floorplanning based on B* tree and fast simulated annealing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 4, pp. 637–650, Apr. 2006.
[24] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.
[25] *GSRC Benchmark*. Accessed: May 26, 2007. [Online]. Available: http://vlsicad.eecs.umich.edu/BK/GSRCbench/HARD/

ZHIPENG HUANG received the B.Sc. degree in information and computing sciences from the College of Mathematics and Computer Science, Fuzhou University, Fuzhou, China, in 2015, and the M.Sc. degree in applied mathematics from the Center for Discrete Mathematics and Theoretical Computer Science, Fuzhou University, in 2018. He is currently pursuing the Ph.D. degree with the Center for Discrete Mathematics and Theoretical Computer Science, Fuzhou University.

His research interests include optimization theory and applications, and VLSI placement.

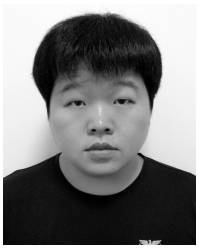
ZHIFENG LIN received the B.Sc. degree in computer science from the College of Mathematics and Informatics, Fujian Normal University, Fuzhou, China, in 2016, and the M.Sc. degree in software engineering from the College of Mathematics and Computer Science, Fuzhou University, in 2019, where he is currently pursuing the Ph.D. degree.

His research interest is FPGA placement.



ZIRAN ZHU received the B.Sc. degree in computer science from the College of Mathematics and Computer Science, Fuzhou University, Fuzhou, China, in 2015. He is currently pursuing the Ph.D. degree with the Center for Discrete Mathematics and Theoretical Computer Science, Fuzhou University.

His research interest includes VLSI placement and routing.



JIANLI CHEN received the B.Sc. degree in information and computing sciences, the M.Sc. degree in computer application technology, and the Ph.D. degree in applied mathematics from Fuzhou University, Fuzhou, China, in 2007, 2009, and 2012, respectively.

He is currently with the State Key Laboratory of ASIC and System, Fudan University, and the Center for Discrete Mathematics and Theoretical Computer Science, Fuzhou University. His research interests include optimization theory and applications, and optimization methods for VLSI physical design automation.

Dr. Chen has received two Best Paper Awards from the International Forum on Operations Research and Control Theory, in 2011 and the Design Automation Conference, in 2017 and the Best Paper Award Candidate from the International Conference on Computer–Aided Design (ICCAD), in 2018. He and his group were a recipient of the First Place Award at the CAD Contest at ICCAD, in 2017, 2018, and 2019, respectively. He has also received the Distinguished Young Scholars Foundation of Fujian Educational Committee, in 2016, and the China Computer Federation Integrated Circuit Early Career Award, in 2018. He has served as a design automation technical committee of the IEEE Council on Electronic Design Automation (CEDA), since January 2018.