Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2014, Article ID 809642, 15 pages
http://dx.doi.org/10.1155/2014/809642

Research Article

Optimal Solution for VLSI Physical Design Automation Using Hybrid Genetic Algorithm

I. Hameem Shanavas[1,2] and R. K. Gnanamurthy[3]

[1] *Anna University, Chennai, India*
[2] *MVJ College of Engineering, Bangalore 560067, India*
[3] *SKP Engineering College, Tiruvannamalai, Tamilnadu 606611, India*

Correspondence should be addressed to I. Hameem Shanavas; hameemshan@gmail.com

Received 30 July 2013; Revised 15 November 2013; Accepted 17 November 2013; Published 9 April 2014

Academic Editor: Hamid Reza Karimi

In Optimization of VLSI Physical Design, area minimization and interconnect length minimization is an important objective in physical design automation of very large scale integration chips. The objective of minimizing the area and interconnect length would scale down the size of integrated chips. To meet the above objective, it is necessary to find an optimal solution for physical design components like partitioning, floorplanning, placement, and routing. This work helps to perform the optimization of the benchmark circuits with the above said components of physical design using hierarchical approach of evolutionary algorithms. The goal of minimizing the delay in partitioning, minimizing the silicon area in floorplanning, minimizing the layout area in placement, minimizing the wirelength in routing has indefinite influence on other criteria like power, clock, speed, cost, and so forth. Hybrid evolutionary algorithm is applied on each of its phases to achieve the objective. Because evolutionary algorithm that includes one or many local search steps within its evolutionary cycles to obtain the minimization of area and interconnect length. This approach combines a hierarchical design like genetic algorithm and simulated annealing to attain the objective. This hybrid approach can quickly produce optimal solutions for the popular benchmarks.

1. Introduction

Physical design automation has been an active area of research for atleast three decades. The main reason is that physical design of chips has become a crucial and critical design task today due to the enormous increase of system complexity and the future advances of electronic circuit design and fabrication. Most commonly used high-level synthesis tools allow the designers to automatically generate huge systems simply by just changing a few lines of code in the functional specification. Nowadays, the open source codes simulated in open source software can automatically be converted to hardware description codes, but the automatically generated codes are not optimized ones. Synthesis and simulation tools often cannot hold with the complexity of the entire system under development. Every time designers want to concentrate on typical parts of a system to upgrade the speed of the design cycle. Thus the present state-of-the-art design technology requires a better solution for the system with fast and effective optimization [1]. Moreover, fabrication and packing technology makes the demand for increasing smaller feature sizes and augmenting the die dimensions possible to allow a circuit for accommodating several millions of transistors; however, logical circuits are restricted in their size and in the number of external pin connections.

So the technology requires partitioning of a system into manageable components by arranging the circuit blocks without wasting empty spaces. The direct implementation of large circuit without going for optimization will occupy large area. Hence the large circuit is necessary to split into small subcircuits. This will minimize the area of the manageable system and the complexity of the large system. When the circuit is partitioned, the connection between two modules or say partitions should be minimum. It is a design task by applying a hierarchical algorithmic approach to solve typical combinatorial optimization problems like dividing a large

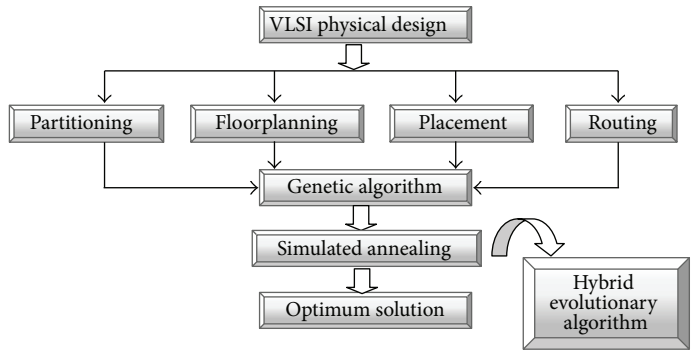


Figure 1: Design flow for the proposed approach.

circuit system into smaller pieces. Figure 1 shows the design flow for the proposed approach.

The method of finding block positions and shapes with minimizing the area objective is referred to as floorplanning. The input to the floorplanning is the output of system partitioning and design entry. Floorplanning paves the way to predict the interconnect delay by estimating the interconnect length. This is achieved because both interconnect delay and gate delay decrease as feature size of the circuit chips is scaled down—but at different rates. The goals of floorplanning are to (a) arrange the blocks on a chip, (b) decide the location of input and output pads, (c) decide the location and number of the power pads (d) decide the type of power distribution, and (e) decide the location and type of clock distribution.

Placement is much more suited to automation than floorplanning. The goal of a placement tool is to arrange all the logic cells with the flexible blocks on a chip. Ideally, the objectives of placement are to (a) minimize all the critical net delays, (b) make the chips as dense as possible, (c) guarantee the router can complete the routing step, (d) minimize power dissipation, and (e) minimize cross-talk between signals. The most commonly used objectives are (a) minimizing the total estimated interconnect length, (b) meeting the timing requirement for critical nets, and (c) minimizing the interconnect congestion.

Once the floorplanning of the chip and the logic cells within the flexible blocks placement are completed, then it is time to make the connection by routing the chip. This is still a hard problem that is made easier by dividing into smaller problems. Routing is usually split into global routing followed by the detailed routing. Global routing is not allowed to finalize the connections; instead it just plans the connections to achieve in a better way. There are two types of areas to global route: one inside the flexible blocks and another between the blocks. The global routing step determines the channels to be used for each interconnect. Using this information the detailed router decides the exact location and layers for each interconnect. The objectives are to minimize the total interconnect length and area and minimize the delay of critical paths [2]. Figure 15 shows the overall area minimized using hybrid evolutionary algorithm.

When the physical design components like partitioning, floorplanning, placement, and routing are combined and optimized in terms of area, then the cost increasing criteria like power and clock speed of each module can be controlled, and these subobjective criteria can also be optimized to a further extent. In the last three decades many interchanging methods have been used which also resulted in local optimum solutions. And later some of the mathematical approaches were also introduced with some heuristics models which resulted in better result but they have their own advantage and disadvantage. Since lots of solutions are possible for this kind of problem, hence stochastic optimization techniques are commonly utilized. Till today many techniques have been proposed like global search algorithm (GSA) which combines the local search algorithm (LSA) to produce a better result.

Global optimization technique like genetic algorithm (GA) which captured the context of generation from biological system had been used for physical design problems like circuit partitioning, floorplanning, placement, and routing. Genetic algorithm has been applied to several problems, which are based on graph because the genetic analogy can be most easily applied to any kind of problems. Lots of researchers have proposed their theories to minimize the feature size of the circuit using GA. Theodore Manikas and James Cain proposed that GA requires more memory but it takes less time than simulated annealing [3]. Sipakoulis et al. confessed that number of enhancements like crossover operator mutation or choosing different fitness functions can still be made to achieve optimal solutions [4]. This means that theory of GA still provides chances for new developments that can help in finding new optimal solutions for physical design problems. This work proposes hybrid evolutionary algorithm to solve the graph physical design component problems. This method includes several genetic algorithm features, namely, selecting population, performing crossover of the selected chromosomes, and if necessary mutation to get better stable solutions. This work tried to hybrid two evolutionary algorithms like genetic algorithm and simulated annealing to overcome the disadvantage of one another. Such type of algorithms with general iterative heuristic approach are called hybrid evolutionary algorithms or memetic algorithm in common.

This work addresses the problem of circuit partitioning with the objective of reducing delay, circuit floorplanning

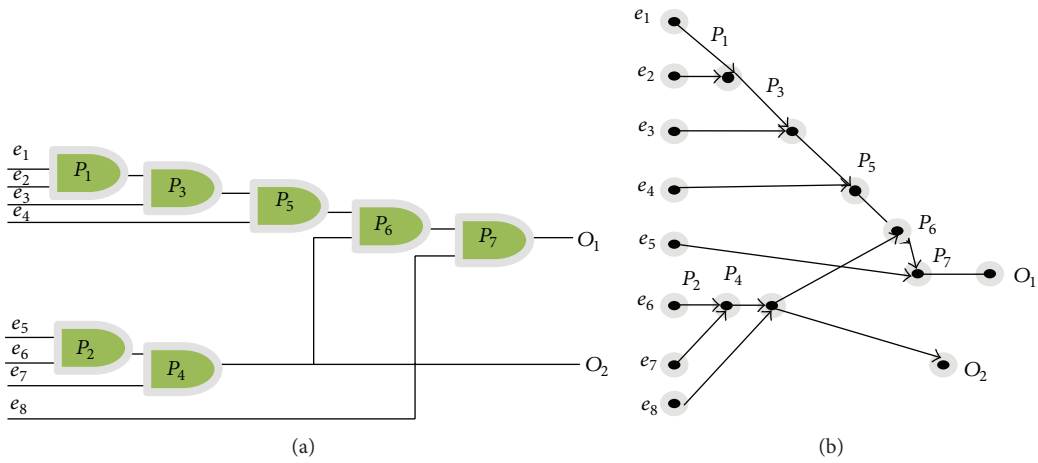


FIGURE 2: (a) Model circuit, (b) graphical representation of circuit.

with the objective of reducing area, placement with the objective of minimizing the layout area, and routing with the objective of minimizing the interconnect length. The main objective of area optimization and interconnect length reduction can be achieved by incorporating hybrid evolutionary algorithm (HEA) in VLSI physical design components.

2. Graphical Representation of Physical Design Components

2.1. Partitioning. Circuit partition will reduce big circuits into small subcircuits and result in a better routing area for the layout. Circuit partitioning problem belongs to the class of NP-hard optimization problems [5]. To measure connectivity, it is necessary to get help from the mathematics of graph theory. Figure 2 states this problem can be considered as a graph partitioning problem where each modules (gates, logic cells, etc.) are taken as vertices (nodes or points) and the connection between two logic cells represents the edges [6].

The algorithm starts with n gates placed on the graph as n vertex, and an initial population has to be chosen as the different permutations of various vertices of the given graph. Given is an unweighted connected graph $G = (V, E)$ on set of vertices V and edges E. Let $n \geq 2$ be a given integer and find a partition $P_1, P_2, P_3, \ldots, P_j$ set of vertices V such that

(i) $G_i = (V_i, E_i)$, for all values $i = 1, 2, 3, \ldots, k$, are connected.

2.2. Floorplanning. A module B is a rectangle of height H_B, width W_B, and area A_B. A super module consists of several modules, also called a subfloorplan. A floorplan for n modules consists of an enveloping rectangle R subdivided by horizontal lines and vertical lines into n nonoverlapping rectangles. Each rectangle must be large enough to accommodate the module assigned to it. In the given problem, a set of hard modules and outline constraints are provided. The modules inside the given outline have freedom to move [7, 8]. A feasible packing is in the first quadrant such that all the modules inside the outline should not duplicate and overlap each other. The objective is to construct a feasible floorplan R such that the total area of the floorplan R is minimized and simultaneously satisfies floorplanning constraint. Given is a set of modules to a specified number of cluster satisfying the predescribed properties. To solve the floorplanning problem, first construct a network graph, and then run the given algorithm to get the solution. The graph consists of two kinds of vertices like horizontal and vertical. The network graph $G = (V, E)$ has to be constructed. "$*$" represents vertical slicing of blocks. "+" represents horizontal slicing of blocks.

2.3. Placement. To position the components of the circuit in general-cell and standard-cell placement, such that the layout area is minimized, the area used here comprises the area used by the circuit components and the area needed for wiring the circuit components. The placement problem can be mainly classified into two, one dimensional packing problem and the other connection cost optimization problem. The packing problem is concerned about fitting a number of cells with different sizes and shapes tightly into a rectangular chip. Given is a set of modules $M_1, M_2, \ldots, M_n$ and set of k interconnects $N_1, N_2, \ldots, N_k$. When meeting the objective of the placement, it can also help to obtain the nonoverlapping package of all the modules which achieves some optimization objective such as minimizing the area of package and the interconnection length as shown in Figure 5.

Horizontal constraint is as follows.

If $(A, B) = (\ldots x \ldots y \ldots, x \ldots y \ldots)$ block y is at the right side of the block x.

Vertical constraint is as follows.

If $(A, B) = (\ldots x \ldots y \ldots, y \ldots x \ldots)$ block y is at the below side of the block x.

(1) Consider $V = \{S_h\} \cup \{T_h\} \cup \{y_i \mid i = 1, \ldots, M\}$, where y_i corresponds to the block, S_h is the source node representing the left boundary, and T_h is the target node representing the right boundary.

(2) Consider $E = \{(S_h, y_i) \mid i = 1, \ldots, M\} \cup \{(y_i, T_h) \mid i = 1, \ldots, N\} \cup \{(y_i, y_j) \mid \ldots y_i \ldots y_j\}$.

If existing edge (y_i, y_{i+1}), edge (y_{i+1}, y_{i+2}), and edge (y_i, y_{i+2}), then (y_i, y_{i+2}) is omitted.

(3) Vertex weight equals the width of the block y_i but zero for S_h and T_h, similarly to the vertical constraint graph (VGH) as shown in Figure 6.

Vertical constraint graph $G_v(V, E)$ is constructed for Figure 5 using "above" constraint and the height of each block. The corresponding constraint graphs $G_h(V, E)$ and $G_v(V, E)$ are as shown in Figures 6 and 7. Both $G_h(V, E)$ and $G_v(V, E)$ are vertex-weighted acyclic graphs so longest path algorithm can be applied to find the x and y coordinates of each block. The coordinates of the block coordinate of the lower left corner of the block.

Based on "left of" constraint of (A, B), a directed and vertex-weighted graph $G_h(V, E)$ (V: vertex set, E: edge set), called the horizontal-constraint graph (HCG), is constructed.

2.4. Routing. The classical approach in routing is to construct an initial solution by using constructive heuristic algorithms. A final solution is then produced by using iterative improvement techniques. A small modification is usually accepted if that makes reduction in cost; otherwise, it will be rejected. Constructive heuristic algorithms produce an initial solution from scratch. It takes a very small amount of computation time compared to iterative improvement algorithms and provides a good starting point for them (SM91). However, the solution generated by constructive algorithms may be far from optimal. Thus, an iterative improvement algorithm is performed next to improving the solution.

Although iterative improvement algorithms can produce a good final solution, the computation time of such algorithms is also large. Therefore, a hierarchical approach in the form of multilevel clustering is utilized to reduce the complexity of the search space. A bottom-up technique gradually clusters cells at several levels of the hierarchy. At the top level a genetic algorithm is applied where several good initial solutions are injected in to the population.

A local search technique with dynamic hill climbing capability is applied to the chromosomes to enhance their quality. The system tackles some of the hard constraints imposed on the problem with intermediate relaxation mechanism to further enhance the solution quality.

This problem is a particular example of graph partitioning problem. In general algorithms like exact and approximation run in polynomial time but do not exist for graph partitioning problems. This makes the necessity to solve the problem using heuristic algorithms. Genetic algorithm is a heuristic technique and the best choice that seeks to imitate the behavior of biological reproduction and its capability to collectively solve the given problem. GA can provide several alternative solutions to the optimization problem, which are considered as individuals in a population. These solutions are coded as binary strings, called chromosomes. The initial population is constructed randomly. These individuals are evaluated using partitioning by specific fitness function. GA then uses these individuals to produce a new generation of hopefully better solutions. In each generation, two of the individuals are selected probabilistically as parents, with the selection probability proportional to their fitness. Crossover is performed on these individuals to generate two new individuals, called offspring, by exchanging parts of their structure. Thus each offspring inherits a combination of features from both parents. The next step is mutation. An incremental change is made to each member of the population, with a small probability. This ensures that GA can explore new features that may not be in the population yet. It makes the entire search space reachable, despite the finite population size. The basic foundation of the algorithm is to represent each vertex in the graph as a location that can represent a logic gate and a connection is represented by an edge.

3. Global Optimization Using GA

Genetic algorithms are optimization strategies that imitate the biological evolution process. A population of individuals representing different problem solutions is subjected to genetic operators, such as selection, crossover, and mutation, that are derived from the model of evolution. Using these operators the individuals are steadily improved over many generations and eventually the best individual resulting from this process is presented as the best solution to the problem.

Consider the graph $G = (V, E)$ with vertex $|v| = u$ and in integer $1 < k < n/4$. Initialize a randomly generated population P of k elements. Population P has 1 to k elements. Assume each parent p_1 to p_k belong to the population P. Perform two point crossover for p_a and p_b from population P using the fitness function $(f) = k \cdot M(p)/n$, where $M(p)$ is the number of node of partition with maximum cardinality among n partitions. Assume $P_a(I)$ and $P_b(I)$ are the children from p_a and p_b, respectively. If $P_a(I)$ has not satisfied the fitness ($P_a(I)$ is not in I) then choose p_a randomly from $j > i$. Swap $P_a(i)$ and $P_a(j)$. Copy the first element k elements of p_a in q_1, q_3. If $P_b(I)$ has not satisfied the fitness ($P_b(I)$ is not in I) then choose p_b randomly from $h > k$. Swap $P_b(h)$ and $P_b(i)$. Copy the first element k elements of p_b in q_2, q_4. Create two vectors L, L' with $2(n - k)$ elements. If (j mod 2 = 1) then $L(i) = P_a[k + (j + 1)/2]$ and $L'(i) = P_b[k + (j + 1)/2]$, else $L(j) = P_a[k + (j + 1)/2]$ and $L'(j) = P_b[k + (j + 1)/2]$. Check the fitness of $L(i)$, $L'(i)$, $L(j)$, and $L'(j)$. If $L(i)$ is not in q_1, then copy $L'(i)$ in q_1 and $L(i)$ in q_2. If $L(j)$ is not in q_3, then copy $L'(j)$ in q_3 and $L(j)$ in q_4. Repeat the process again with $L(i)$ and $L(j)$, $L'(i)$ and $L'(j)$ to get new offspring. The new offsprings can have more fitness value or less fitness value depending upon the parents. Less fitness offspring can be discarded then to reduce the number of cycles. In this work, the pure genetic algorithm is combined with simulated annealing to produce the optimal result. The algorithm starts with the initial random population generation. It is essential to set the initial population, number of generation, crossover type, and mutation rate. First step of the genetic algorithm starts with the selection process; the selection process is based on the fitness function through which the chromosomes were selected from the crossover. Crossover is the reference point

for the next generation population. The crossover technique used in genetic algorithm is one-point crossover, two-point crossover, cut and splice crossover, uniform crossover, half uniform crossover, and so forth, depending upon the necessity. After crossover the mutation process, to maintain the genetic diversity from one generation to next generation. In this mutation the genetic sequence will be changed from its original sequence by generating a random variable for each bit sequence. After the mutation offspring with fitness are placed in the new population for further iteration. The next step is to apply the local optimization algorithm in between this genetic algorithm as told before; the local optimization is applied in three ways which are mentioned below: (a) before the crossover, (b) after the crossover, and (c) before and after the crossover.

Exhaustive Hybridization. Few solutions are selected from the final generation and improved using local search. Figure 16 shows simulated results for final generation.

Intermediate Hybridization. After a predetermined number of iterations by GA, local search is applied to few random individuals. This is done to have a better solution than the local maxima. This work deals with intermediate memetic algorithm.

3.1. Creation of Initial Population. The initial population is constructed from randomly created routing structures, that is, individuals. First, each of these individuals is assigned a random initial row number y_{ind}. Let $S = \{s_1, \ldots, s_i, \ldots, s_k\}$ be the set of all pins of the channel which are not connected yet and let $T = \{t_1, \ldots, t_j, \ldots, t_l\}$ be the set of all pins having at least one connection to other pin. Initially $T = 0$. A pin $S_i \in S$ is chosen randomly among all elements in S. If T contains pins $\{t_u, \ldots, t_j, \ldots, t_v\}$ (with $1 \le u < v \le l$) of the same net, a pin t_j is randomly selected among them. Otherwise, a second pin of the same net is randomly chosen from S and transferred into T. Both pins (s_i, t_j) are connected with a so-called random routing. Then s_i is transferred into T. The process continues with the next random selection of $s_i \in S$ until $S = 0$. The creation of the initial population is finished when the number of completely routed channels is equal to the population size $|P_c|$. As a consequence of our strategy, these initial individuals are quite different from each other and scattered all over the search space.

3.2. Populations and Chromosomes. In GA based optimizations a set of trial solutions are assembled as a population. The parameter set representing each trial solution or individual is coded to form a string or chromosome and each individual is assigned a fitness value by evaluation of the objective function. The objective function is to only link between the GA optimizer and the physical problem.

3.3. Calculation of Fitness. The fitness of the individual in partitioning is based on the delay of the module. The fitness of the individual is given by weighted evaluations of maximum delay (D). D_a identifies a particular subgraph, D_m is a predetermined maximum value, and D_f is the weighting factor. Delay can be measured using the difference between final time and initial time. The sum of the weighting factors equals one. The complete fitness function for partitioning is given in the following:

$$D_p = \left\{ \frac{D_a}{D_m} > 1, \frac{D_a}{D_m} \le 1, \frac{D_a}{D_m} * D_f \right\}, \tag{1}$$

$$\text{Total Delay} = \sum_{P=1}^{M} Dp. \tag{2}$$

Assuming an individual is fully feasible and meets constraints, the value of $D_p \le 1$, with smaller values being better.

At the beginning, a set of Polish expressions is given for floorplanning. It is denoted as P, randomly generated expression to compose a population. The fitness for floorplanning of the individual is based on the area of the module. Area of a block can be calculated by the general formula $A = LW$, where L stands for length of the module and W stands for width of the module:

$$\text{Total Area} = \sum_{f=1}^{M} Af. \tag{3}$$

The fitness of the individual is given by weighted evaluations of maximum area (A). A_a identifies a particular subgraph, A_m is a predetermined maximum value, and A_f is the weighting factor. The sum of the weighting factors equals one. The complete fitness function floorplanning is given in the following:

$$G_f = \left\{ \frac{A_a}{A_m} > 1, \frac{A_a}{A_m} \le 1, \frac{A_a}{A_m} * A_f \right\}. \tag{4}$$

The fitness $F(p)$ of each individual $p \in P$ is calculated to assess the quality of its routing structure relative to the rest of the population P. The selection of the mates for crossover and the selection of individuals which are transferred into the next generation are based on these fitness values. First, two functions $F_1(p)$ and $F_2(p)$ are calculated for each individual $p \in P$ according to

$$F_1(p) = \frac{1}{n_{\text{row}}}, \tag{5}$$

where n_{row} = number of rows of p, and

$$F_2(p) = \frac{1}{\sum_{i=1}^{n_{\text{ind}}} \left(l_{\text{acc}}(i) + a * l_{\text{opp}}(i) \right) + b * v_{\text{ind}}}, \tag{6}$$

where $l_{\text{acc}}(i)$ = net length of net i of net segments according to the preferred direction of the layer, $l_{\text{opp}}(i)$ = net length of net i of net segments opposite to the preferred direction of the layer, a = cost factor for the preferred direction, n_{ind} = number of nets of individual p, v_{ind} = number of vias of individual p, and b = cost factor for vias.

The final fitness $F(p)$ is derived from $F_1(p)$ and $F_2(p)$ in such a way that the area minimization, that is, the number

of rows, always predominates the net length and the number of vias. After the evaluation of $F(p)$ for all individuals of the population P these values are scaled linearly as described in order to control the variance of the fitness in the population.

In placement the cells present in the module are connected by wire. The estimation of interconnect length required for connection is calculated by

$$I_L = \sum_{I>j} w_{i,j} \left(\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2 \right), \tag{7}$$

where $w_{i,j}$ is the weight of the connection between cell x and y, $(x_i - x_j)$ is the distance between two cells in X direction, and $(y_i - y_j)$ is the distance between two cells in Y direction.

Interconnect length of each net in the circuit is estimated during Steiner tree and then total Interconnect length is computed by adding the individual estimates:

$$\text{Cost} = \sum_{L=1}^{M} I_L, \tag{8}$$

where I_L is the interconnect length estimation for net i and M denotes total number of nets in circuit.

3.4. Parents. Following this initialization process, pairs of individuals are selected (with replacement) from the population in a probabilistic manner weighted by their relative fitness and designated as parents.

3.5. Children. A pair of offspring, or children, are then generated from the selected pair of parents by the application of simple stochastic operators. The principle operators are crossover and mutation. Crossover occurs with a probability of pcross (typ. 0.6–0.8) and involves the random selection of a crossover site and combining the two parent's genetic information. The two children produced share the characteristics of the parents as a result of these recombination operators. Other recombination operators are sometimes used, but crossover is the most important. Recombination (e.g., crossover) and selection are the principle way that evolution occurs in a GA optimization.

3.6. Mutation. Mutation introduces new and unexplored points into the GA optimizer's search domain. Mutation randomly changes the genetic makeup of the population. Mutation is much less important than recombination and occurs with a probability pmutation (typ. 0.05) which is much less than pcross.

3.7. New Generation. Reproduction consisting of selection, recombination, and mutation continues until a new generation is created to replace the original generation. Highly fit individuals, or more precisely highly fit characteristics, produce more copies of themselves in subsequent generation resulting in a general drift of the population as a whole towards an optimal solution point. The process can be terminated in several ways: threshold on the best individual (i.e., the process stops when an individual has an error less than some amount E), number of generations exceeds a preselected value, or some other appropriate criteria. A simple genetic algorithm must be able to perform five basic tasks: encode the solution parameters in the form of chromosomes, initialize a starting point population, evaluate and assign fitness values to individuals in the population, perform reproduction through the fitness weighted selection of individuals from the population, and perform recombination and mutation to produce members of the next generation [8–14].

4. Local Optimization Using SA

After a prescribed number of iterations by evolutionary algorithm local search algorithm is applied to few random individuals to have better solution. But simulated annealing algorithm is not a local search algorithm. Local search methods are iterative algorithms that tend to enhance solution by stepwise improvement and make an attempt to reach optimum solutions. SA is being used in this work. More often results in suboptimal solutions by trapping themselves in local minima/maxima. The simplest form of local search is repetitively flipping elements in a solution resulting in a gain in the objective function. Eventually local minima will be reached, whereby flipping any element in the solution will result in loss of object. Although these algorithms are simple, there have been many complex improvements for CAD tools which involve large dynamic memory and linked list usages. For refining the solution obtained by GA, the local search (LS) is applied. This can be used before crossover or after crossover; it can also be used for parents selection and used before or after mutation to increase the number of fitness variables (Algorithm 1).

5. Optimization by Simulated Annealing

Simulated annealing algorithm is applied for local search process since SA is not a local search algorithm. Here simulated annealing method is performed on finally generated offspring to improve the fitness. This method is called intermediate MA.

Simulated annealing is a stochastic computational method for finding global extrema to large optimization problems. It was first proposed as an optimization technique by Kirkpatrick et al. in 1983 [15] and Cerny in 1984 [16]. The optimization problem can be formulated by describing a discrete set of configurations (i.e., parameter values) and the objective function to be optimized. The problem is then to find a vector that is optimal. The optimization algorithm is based on a physical annealing analogy. Physical annealing is a process in which a solid is first heated until all particles are randomly arranged in a liquid state, followed by a slow cooling process. At each (cooling) temperature enough time is spent for the solid to reach thermal equilibrium, where energy levels follow a Boltzmann distribution. As temperature decreases the probability tends to concentrate on low energy states. Care must be taken to reach thermal equilibrium prior to decreasing the temperature. At thermal

```
Begin
   Initialize population P;
   For i := 1 To size of (P) Do
         Individual := P_i;
         Individual := Local_Search(Individual);
   Repeat Until (terminate = True) Do
      For i := 1 To #recombination Do
         Select two parents i_a, i_b ∈ P randomly;
         i_c := Recombination(i_a, i_b);
         i_c := Local Search(i_c);
   Add individual i_c to P;
   P := Select (P);
   If (P) converged Then
         For i := 1 To size of (P), i != index(Best) Do
         Individual := P_i
   Individual := Local_Search(Mutate(Individual));
End
```

Algorithm 1

equilibrium, the probability that a system is in a macroscopic configuration with energy is given by the Boltzmann distribution. The behavior of a system of particles can be simulated using a stochastic relaxation technique developed by Metropolis et al. [17]. The candidate configuration for the time is generated randomly. The new candidate is accepted or rejected based on the difference between the energies associated with states. The condition to be accepted is determined by

$$p = \frac{p_r}{p_q} = \frac{\exp\left(-E_j - E_i\right)}{K_t} > 1. \tag{9}$$

Given a current state i of the solid with energy level E_i, generate a subsequent state j randomly (by small perturbation).

Let E_j be the energy level at state j.

(i) If $E_j - E_i \leq 0$, then accept state j as the current state.

(ii) If $E_j - E_i > 0$, then accept state j with the probability $\exp(-E_j - E_i)/K_t$.

K_t where k is the Boltzmann constant.

One feature of the Metropolis way of simulated annealing algorithm is that a transition out of a local minimum is always possible at nonzero temperature. Another evenly interesting property of the algorithm is that it performs a kind of adaptive divide and conquer approach. Gross features of the system appear at higher temperatures; fine features develop at lower temperatures. For this application, it used the implementation by Ingber [18]. Each solution corresponds to a state of the system. Cost corresponds to the energy level. Neighborhood corresponds to a set of subsequent states that the current state can reach. Control parameter corresponds to temperature.

5.1. Partitioning Based on Simulated Annealing. The basic procedure in simulated annealing is to start with an initial partitioning and accept all perturbations or moves which result in a reduction in cost. Moves that result in a cost increase are accepted. The probability of accepting such a move decreasing with the increase in cost and also decreasing in later stages of the algorithm is given in (11). A parameter T, called the temperature, is used to control the acceptance probability of the cost-increasing moves. Simulated annealing algorithm for partitioning the modules will be described here. The cells are partitioned using simulated annealing so as to minimize the estimated interconnect length. There are two methods for generating new configurations from the current configuration [19]. Either a cell is chosen randomly and placed in a random location on the chip or two cells are selected randomly and interchanged. The performance of the algorithm was observed to depend upon r, the ratio of displacements to interchanges. Experimentally, r is chosen between 3 and 8. A temperature-dependent range limiter is used to limit the distance over which a cell can move. Initially, the span of the range limiter is twice the span of the chip. In other words, there is no effective range limiter for the high temperature range. The span decreases logarithmically with the temperature. Temperature span is given in the following:

$$\begin{aligned} L_{W_V}(T) &= L_{W_V}(T_i)\left[\frac{\log T}{\log T_i}\right], \\ L_{W_H}(T) &= L_{W_H}(T_i)\left[\frac{\log T}{\log T_i}\right], \end{aligned} \tag{10}$$

where T is the current temperature, T_i is the initial temperature, and $L_{W_V}(T_i)$ and $L_{W_H}(T_i)$ are the initial values of the vertical and horizontal window spans $L_{W_V}(T)$ and $L_{W_H}(T)$, respectively.

The wirelength cost C is estimated using the semiperimeter method, with weighting of critical nets and independent weighting of horizontal and vertical wiring spans for each net:

$$C = \sum_{\text{nets } i} \left[x(i) W_H(i) + y(i) W_V(i)\right], \tag{11}$$

where $x(i)$ and $y(i)$ are the vertical and horizontal spans of the net i's bounding rectangle and $W_H(i)$ and $W_V(i)$ are the weights of the horizontal and vertical wiring spans. When critical nets are assigned a higher weight, the annealing algorithm will try to place the cells interconnected by critical nets close to each other. Independent horizontal and vertical weights give the user the flexibility to prefer connections in one direction over the other. The acceptance probability is given by $\exp(-\Delta C/T)$, where ΔC (i.e., $E_i - E_j$) is the cost increase and T is the current temperature. When the cost increases, or when the temperature decreases, the acceptance probability (9) gets closer to zero. Thus, the acceptance probability $\exp(-\Delta C/T)$ less than random (0, 1) (a random number between 0 and 1) is high when ΔC is small and when T is large. At each temperature, a fixed number of moves per cell is allowed. This number is specified by the user. The higher the maximum number of moves, the better the results obtained. However, the computation time increases rapidly. There is a recommended number of moves per cell as a function of the problem size in. For example, for a 200-cell and 3000-cell circuit, 100 and 700 moves per cell are recommended, respectively.

The annealing process starts at a very high temperature, for example, $T_i = 4{,}000{,}000$, to accept most of the moves. The cooling schedule is represented by $T_i + 1 = a(T)$, where $a(T)$ is the cooling rate parameter and is determined experimentally. In the high and low temperature ranges, the temperature is reduced rapidly (e.g., $a(T) \approx 0.8$). However, in the medium temperature range, the temperature is reduced slowly (e.g., $a(T) \approx 0.95$). The algorithm is terminated when T is very small, for example, when $T < 0.1$. Within each temperature range the number of moves has been built experimentally. Once the number of moves is set, fix the particular move for the remainder of the scheduling.

5.2. Floorplanning Based on Simulated Annealing. This section describes an optimal floorplanning on simulated annealing algorithm. Assume that a set of modules is given and each module can be implemented in a finite number of ways, characterized by its width and height. Some of the important issues in the design of a simulated annealing optimization problem are as follows:

(1) the solution space,

(2) the movement from one solution to another,

(3) the cost evaluation function.

The branch cells correspond to the operands and the internal nodes correspond to the operators of the Polish expression. Figure 3 shows the floorplan module. A binary tree can also be constructed from a Polish expression by using a stack as shown in Figure 4. The simulated annealing algorithm moves from one Polish expression to another. A floorplan may have different slicing tree representations. For example, the tree in Figure 4 represents the given floorplan in Figures 8, 9, and 10. There is a one-to-one correspondence between a floorplan and its normalized Polish expression. But this leads to a larger solution space and some bias towards floorplans with multitree representations, since they have more chances to be visited in the annealing process. Assume three types of movement are defined to move from one floorplan to another. They operate on the Polish expression representation of the floorplan [20].

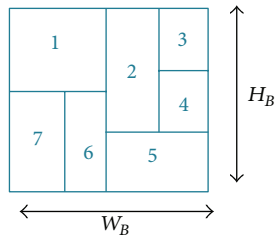
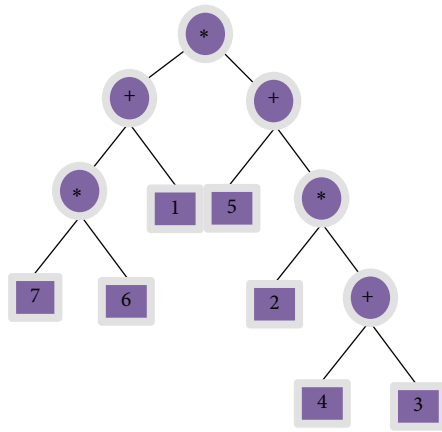


Figure 3: Floorplan module.



Polish expression: 76∗1+43+2∗5+∗

Figure 4: Graph representation of the given floorplan module.

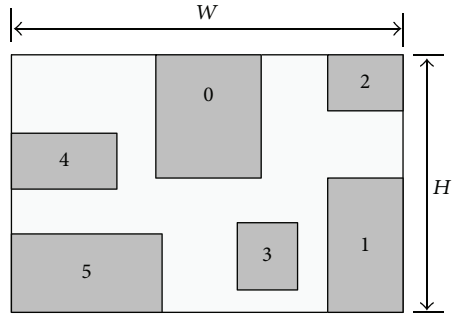


Figure 5: Placement module.

Condition 1. Exchange two operands when there are no other operands in between $(67 * 1 + 34 + 2 * 5 + *)$.

Condition 2. Complement a series of operators between two operands $(67 + 1 * 34 * 2 + 5 * +)$.

Condition 3. Exchange adjacent operand and operator if the resulting expression is a normalized Polish expression $(67 + 43 * +25 * 1 + *)$.

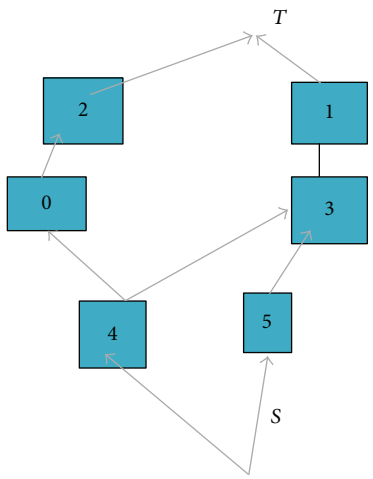


Figure 6: Vertical constraint graph (VGH) for the given block $G_v(V, E)$.

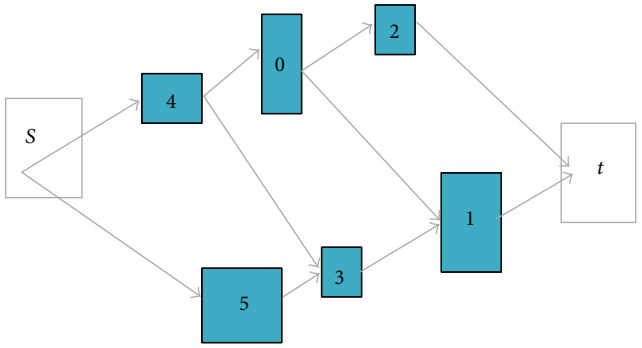


Figure 7: Horizontal constraint graph (HGH) for the given block $G_h(V, E)$.

It is obvious that the movements will generate only normalized Polish expressions. Thus, in effect, the algorithm moves from one floorplan to another. Starting from an arbitrary floorplan, it is possible to visit all the floorplans using the movement. If some floorplans cannot be reached, there is a danger of losing some valid floorplans in the solution. Starting from any floorplan, the modules can move to the floorplan based on the given conditions. The cost function is a function of the floorplan or equivalently the Polish expression. There are two components for the cost function, area and wirelength. The area of a sliceable floorplan can be computed easily using the floorplan sizing algorithm [21]. The wirelength cost can be estimated from the perimeter of the bounding box of each net, assuming that all terminals are located on the center of their module. In general, there is no universally agreed upon method of cost estimation. For simulated annealing, the cost function is best evaluated easily because thousands of solutions need to be examined. Figures 8, 9, and 10 show a series of movements which lead to a solution. We have implemented the exponential function for the accept method.

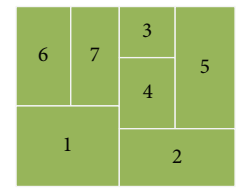


Figure 8: Condition 1 $(67 * 1 + 34 + 2 * 5 + *)$.

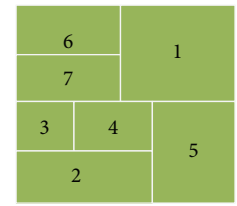


Figure 9: Condition 2 $(67 + 1 * 34 * 2 + 5 * +)$.

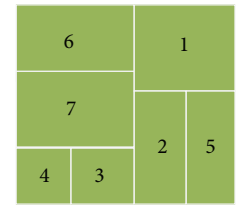


Figure 10: Condition 3 $(67 + 43 * +25 * 1 + *)$.

5.3. Placement Based on Simulated Annealing. Simulated annealing algorithm mimics the annealing process used to gradually cool molten metal to produce high quality metal structures:

(i) initial placement improved by iterative swaps and moves,

(ii) accept swaps if they improve the cost,

(iii) accept swaps that degrade the cost under some probability conditions to prevent the algorithm from being trapped in a local minimum and can reach globally optimal solution given enough time.

The advantage of using SA are open cost function, wirelength cost, and timing cost. Along with the advantage it also has its disadvantage of slowness. The purpose of our algorithm is to find a placement of the standard cells such that the total estimated interconnection cost is minimized. The algorithm for placement if divide into four principal components [22].

5.3.1. Initial Configuration. Initially the circuit decomposed into individual cells and found out the input and output cells for each cell.

Then it starts the annealing procedure by placing the cells on the chip randomly. And finally it calculates the total area of the circuit and places the cells accordingly so that they are placed at equal distances from each other.

Table 1: Partitioning optimization of GA compared with hybrid algorithm.

Circuit	GA			Hybrid		
	Delay (ps)	T (s)	Best (s)	Delay (ps)	T (s)	Best (s)
S1196	396	375	373	301	184	134
S1238	475	397	365	408	187	160
S1494	614	1228	1040	585	616	427
S2091	302	94	32	225	616	16
S3330	571	2096	2074	533	470	994
S5378	587	2687	2686	590	1078	1100

Table 2: Floorplanning optim of GA compared with hybrid algorithm.

Circuit	Number of blocks	GA		Hybrid	
		Wirelength	CPU time	Wirelength	CPU time
xerox	10	33.56	30	32.06	26
Ami33	33	35.44	30	34.39	24
apte	9	25.48	30	24.23	22
Ami49	49	63.55	175	58.33	150
hp	11	28.46	45	27.54	60

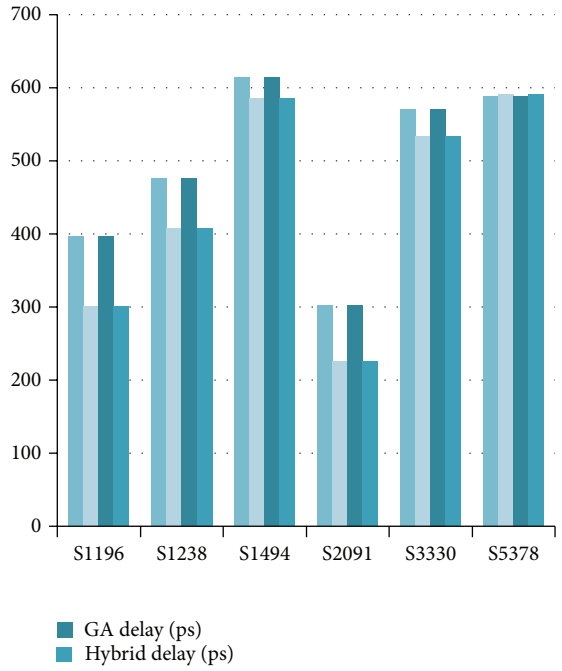


Figure 11: Comparison of GA delay and hybrid delay.

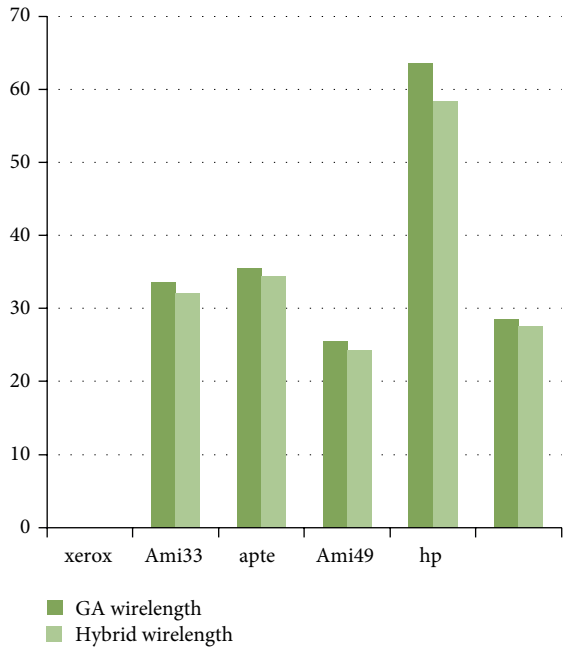


Figure 12: Comparison of GA wirelength and hybrid wirelength.

Since the cells are placed randomly, thus the distances between them and the length of their interconnection will be huge. Next the algorithm uses three different functions to get the optimal placement for the chip.

5.3.2. Move Generation Function. To generate a new possible cell placement, the algorithm uses two strategies:

(a) move a single cell randomly to a new location on the chip,

(b) swap the position of two cells. This algorithm uses both the strategies randomly. 50% of the move generation is done through random move (a) and the rest of the generation is done through swapping (50%).

5.3.3. Cost Function. The cost function in algorithm is comprised of two components:

$$C = C_1 + C_2. \tag{12}$$

Table 3: Placement optim of SA compared with hybrid algorithm.

Circuit	GA			Hybrid		
	Cells	Nets	Final interconnect length	Cells	Nets	Final interconnect length
apte	9	97	590.6	9	77	475.4
xerox	10	203	1038	10	171	1145
hp	11	83	365	11	68	283
Ami33	33	123	278.5	33	112	324
Ami49	49	408	2077	49	368	345

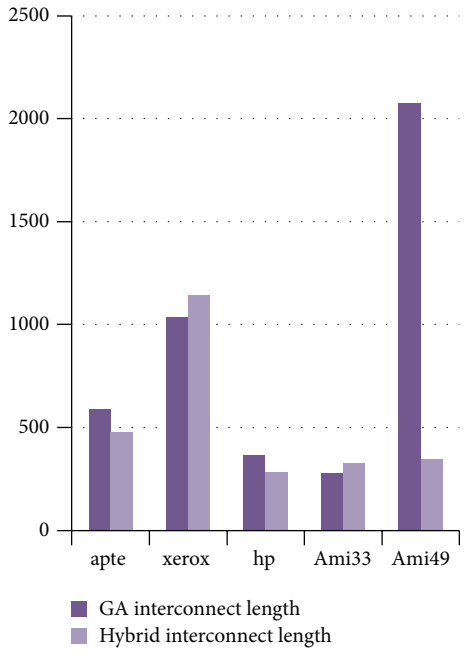


Figure 13: Comparison of GA interconnect length and hybrid.

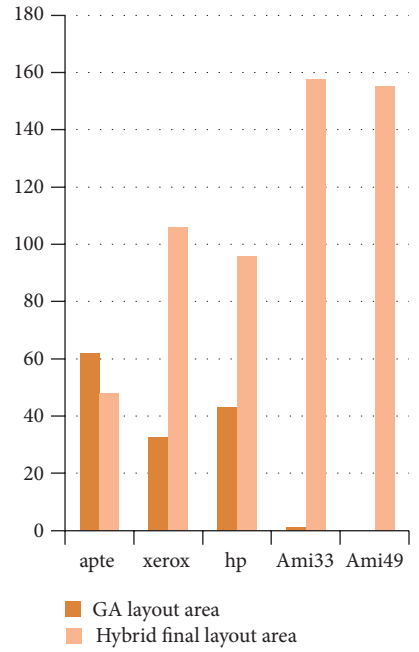


Figure 14: Comparison of GA layout area and hybrid.

C_1 is a measure of the total estimated wirelength. For any cell, we find out the wire-length by calculating the horizontal and the vertical distance between it and its out cell:

$$C_1 = \sum_{i=\text{cell1}}^{n} \sum_{j=\text{out cell}}^{n} \left(d_{hij} + d_{vij}\right), \tag{13}$$

where the summation is taken over all the cells in a circuit. When a cell is swapped it may so happen that two cells overlap with each other. Let Oij indicate the overlap between two cells. Clearly this overlap is undesirable and should be minimized. In order to penalize the overlap severely we square the overlap so that we get larger penalties for overlaps:

$$C_2 = \sum_{i!=i} (Oij)^2. \tag{14}$$

In (14) C_2 denotes the total overlap of a chip. Thus when we generate a new move we calculate the cost function for the newly generated move. If we find that the new move has a cost less than the previous best move, we accept it as the best move. But if we find a solution that is nonoptimal, we do not reject it completely. We define an Accept function which is the probabilistic acceptance function. It determines whether to accept a move or not. We have implemented an exponential function for the accept method. We are accepting a noncost optimal solution because we are giving the annealing schedule a chance to move out of a local minimum which it may have hit. For example, if a certain annealing schedule hits point B (local minima) and if we do not accept a noncost optimal solution, then the annealing cannot reach the global minima. By using the accept function we are giving the annealing schedule a chance to get out of the local minima. As a nature of the accept function used by us, the probability of accepting noncost optimal solution is higher at the beginning of the annealing schedule. As temperature decreases, so does the probability of accepting noncost optimal solutions, since the perturbations of a circuit are higher at higher temperatures than lower temperatures.

5.4. Routing Based on Simulated Annealing. Let p_α and p_β be copies of the mates and p_γ be their descendant.

Table 4: Routing optim of SA compared with hybrid algorithm.

Circuit	GA			Hybrid		
	Cells	Nets	Final layout area	Cells	Nets	Final layout area
apte	9	97	61.8	9	77	48.012
xerox	10	203	32.6	10	171	106.054
hp	11	83	42.9	11	68	95.862
Ami33	33	123	1.23	33	112	157.690
Ami49	49	408	—	49	368	155.24

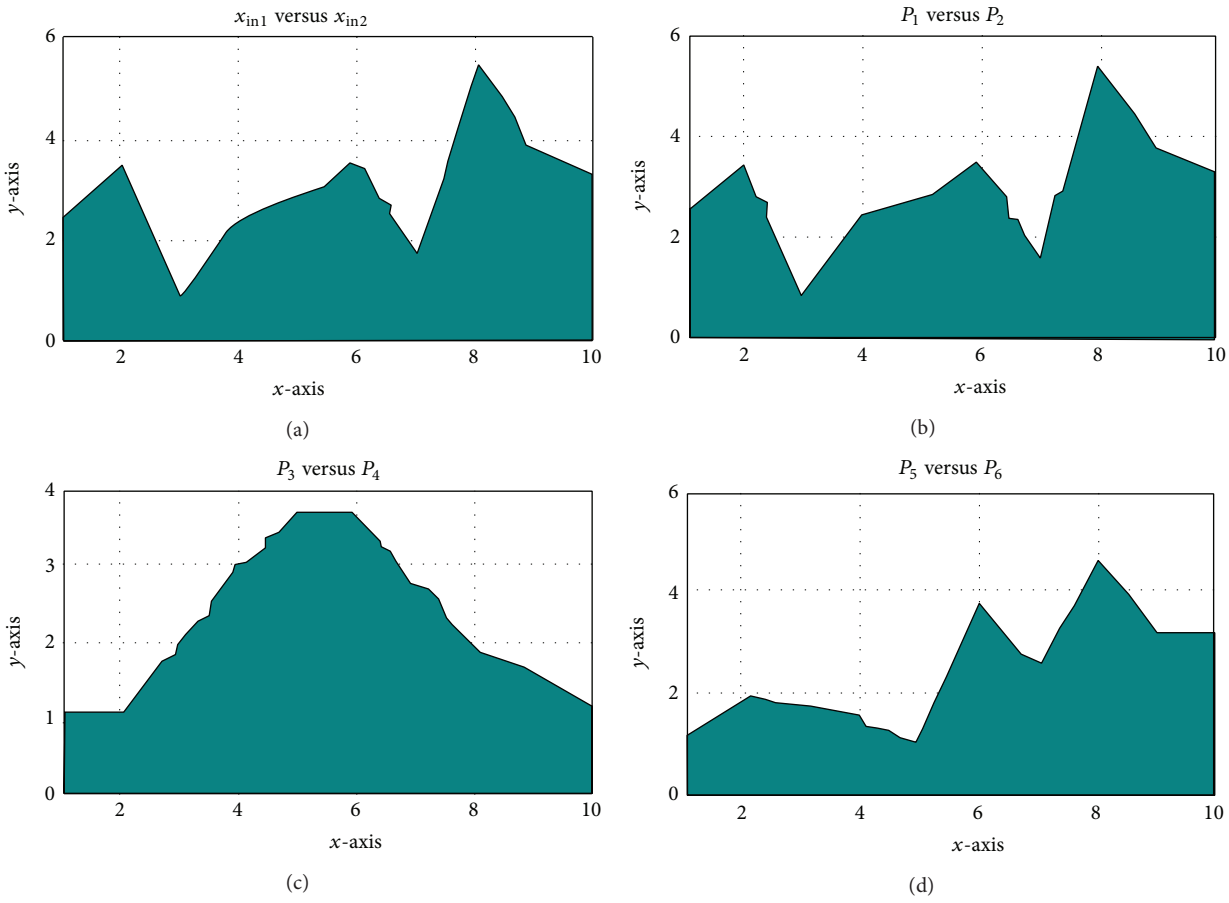


Figure 15: Overall area minimized using hybrid evolutionary algorithm.

First, a cut column x_c is randomly selected with $1 \leq x_c < x_{ind}$, where x_{ind} represents the number of columns of the individuals. The individual $p_\alpha(p_\beta)$ transfers the routing structure to p_γ which is located to the left (right) of the cut column x_c and not touched by x_c. Assume that the part of p_α (or p_β) which has to be transferred into p_γ contains rows not occupied by any horizontal segments. Then the row number of p_α (or p_β) is decremented by deleting this row until no empty row is left. The initial row number $y_{ind\gamma}$ of p_γ is equal to the maximum of ($y_{ind\alpha}$, $y_{ind\beta}$). The mate which now contains fewer rows than p_γ is extended with additional row(s) at random position(s) before transferring its routing structure to p_γ.

The routing of the remaining open connections in p_γ is done in a random order by our random routing strategy. If the random routing of two points does not lead to a connection within a certain number of extension lines, the extension lines are deleted and the channel is extended at a random position y_{add} with $1 \leq y_{add} \leq y_{ind\gamma}$. If the repeated extension of the channel also does not enable a connection, p_γ is deleted entirely and the crossover process starts again with a new random cut column x_c applied to p_α and p_β. This process of creating p_γ is finished with deleting all rows in p_γ that are not used for any horizontal routing segment [23, 24].

Reduction strategy simply chooses the P_c fittest individuals of ($P_c \upsilon P_n$) to survive as P_c into the next generation.

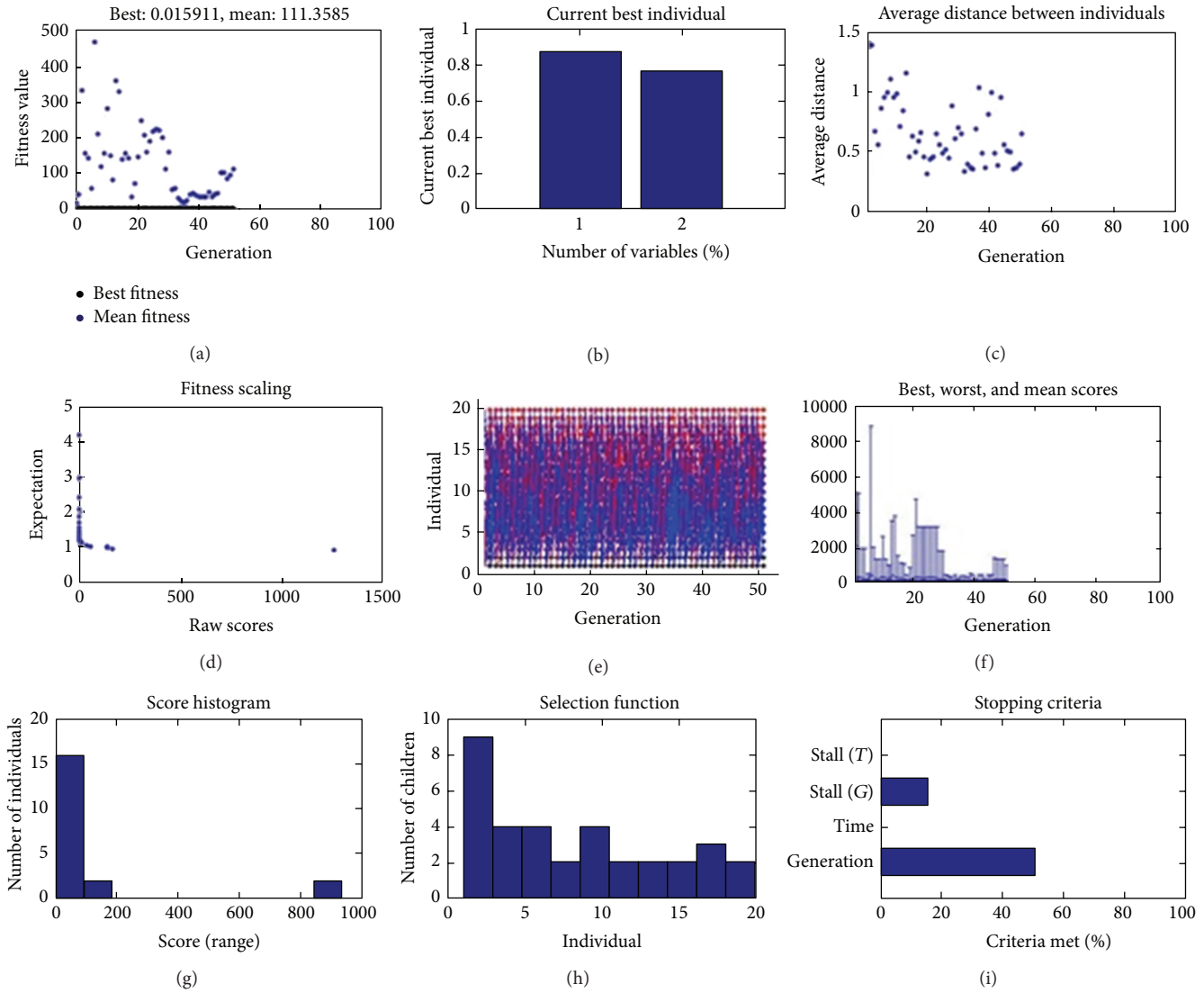


Figure 16: Simulated results for final generation.

The selection strategy is responsible for choosing the mates among the individuals of the population P_c.

According to the terminology of our selection strategy is actually stochastic sampling with replacement. That means any individual $p_i \in p_c$ is selected with a probability

$$F(p_i) = \frac{1}{\sum_{p \in p_c} F(p)}. \quad (15)$$

The two mates needed for one crossover are chosen of each other. An individual may be independently selected any number of times in the same generation.

6. Experimental Results

This work compares the performance of combined physical design automation tool for different benchmarks of physical design components. This iterative heuristic technique involves the combination of GA and SA. This work measures the speed of execution time of all levels on an average of 45% when compared to simple genetic algorithm. The objective of individual physical design components is discussed below as results. The results in this section were obtained by the simulation of each individual element of physical design components using general iterative heuristic approach. The experiments were executed on the Intel Core i3 processor with the clock speed of 3.3 Ghz machine which runs in Windows XP.

6.1. Partitioning. Delay (ps) is the delay of the most critical path. T (s) is the total run time, and best (s) is the execution time in seconds for reaching the best solution. Thus the objective of area minimization can be achieved by reducing the delay in circuit partitioning (see Table 1 and Figure 11).

6.2. Floorplanning. In floorplanning, wirelength and CPU time are compared. This heuristic approach can reduce

the wirelength on an average of 0.5 mm when compared with fast simulated annealing. When the wirelength gets reduced, obviously the area for floorplan will also get reduced (see Table 2 and Figure 12).

6.3. Placement. The initial population is generated to evaluate the fitness function. Based on that fitness parents were selected for the crossover; after this process the normal mutation and inversion operation take place. In addition to this process for each subpopulation local search is applied to refine the fitness of each individual to get the optimal solution. The cells describe the number of elements in the circuits, nets describe the interconnection (see Table 3 and Figure 13).

6.4. Routing. The method is also surprisingly fast, even when compared to tools that perform pattern routing. Improving routing congestion is significant concern for large and dense designs. If routing fails, the design team must modify the placement or possibly increase the chip size to introduce additional routing resources. In fixed-die design, if there is additional space available, the impact of increased routing area will generally be limited to increased wirelength and power consumption. If additional space is not available, routing failure may increase the cost of a design substantially. The results obtained for the popular benchmarks reduce the final interconnect length (see Table 4 and Figure 14).

7. Conclusion

By reducing the wirelength, cost of very large scale integrated chip manufacturing can be reduced. This paper explores the advantage of memetic algorithm which can be proven to be 45% faster than the simple software genetic algorithm by reducing the delay and area in partitioning and floorplanning, respectively, that would indefinitely reduce the wirelength. In hybrid approaches, local search techniques explore the solution space close to the sample points by applying specialized heuristics. When including problem specific knowledge during creation of individual, like in our approach, it is possible to identify unfavourable or redundant partial solutions and consider only the most promising ones. Therefore, each individual in our hybrid genetic algorithms encodes a set of high quality solutions, the best of which is a local optimum. The implementation of multiobjective in the algorithm enables getting the near optimal solution. After a predetermined number of iterations by GA, local search is applied to few random individuals to get the optimal solution by simulated annealing. In the future the performance of the algorithm has to be tested on different benchmarks.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] I. H. Shanavas and R. K. Gnanamurthy, "Wirelength minimization in partitioning and floorplanning using evolutionary algorithms," *VLSI Design*, vol. 2011, Article ID 896241, 9 pages, 2011.

[2] M. J. S. Smith, *Application Specific Integrated Circuits*, Pearson Education, Asia, 2004.

[3] W. Theodore Manikas and T. James Cain, "Genetic algorithms vs simulated annealing: a comparison of approaches for solving the circuit partitioning problem," Tech. Rep. 96-101, The University of Pittsburgh, 1996, http://www.ee.utulsa.edu/~tmanikas / Pubs/gasa-TR-96-101.pdf.

[4] G. Ch. Sipakoulis, I. Karafyllidis, and A. Thanailkis, "Genetic partition and placement for VLSI circuits," in *Proceedings of 6th IEEE conference on Electronics Circuits and Systems*, vol. 3, pp. 1647–1650, 1999.

[5] M. R. Garey and Johnson, *Computers and Intractability: A Guide to Theory of NP-Completeness*, Freeman, San Francisco, Calif, USA.

[6] C. J. Augeri and H. H. Ali, "New graph-based algorithms for partitioning VLSI circuits," in *Proceedings of the IEEE International Symposium on Cirquits and Systems*, vol. 6, pp. V-521–V-524, May 2004.

[7] S. N. Adya and I. L. Markov, "Fixed-outline floorplanning through better local search," in *Proceedings of the IEEE International Conference on: Computer Design: VLSI in Computers and Processors (ICCD '01)*, pp. 328–334, Austin, Tex, USA, September 2001.

[8] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, and S.-W. Wu, "B∗-trees: a new representation for non-slicing floorplans," in *Proceedings of the 37th Design Automation Conference (DAC '00)*, pp. 458–463, June 2000.

[9] M. Tang and X. Yao, "A memetic algorithm for VLSI floor planning," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 37, no. 1, pp. 62–69, 2007.

[10] N. Xu, F. Huang, and Z. Jiang, "A fast algorithm for VLSI building block placement," in *Proceedings of the IMACS Multiconference on "Computational Engineering in Systems Applications" (CESA '06)*, pp. 2157–2160, Beijing, China, October 2006.

[11] M. Burstein and R. Pelavin, "Hierarchical wire routing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 2, no. 4, pp. 223–234, 1983.

[12] A. Cincotti, V. Cuttelo, and M. Pavone, "Graph partitioning using genetic algorithms with OPDX," in *Proceedings of IEEE on World Congress on Computational Intelligence*, pp. 402–406, May 2002.

[13] I. Hameem Shanavas and R. K. Gnanamurthy, "Evolutionary algorithmical approach on VLSI Floorplanning problem," *International Journal of Computer Theory and Engineering*, vol. 1, no. 4, pp. 461–464.

[14] N. Sherwani, *Algorithms for VLSI Physical Design Automation*, Kluwer Academic Publishers, Boston, Mass, USA, 1999.

[15] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[16] V. Cerny, "Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm," *Journal of Optimization Theory and Applications*, vol. 45, no. 1, pp. 41–45, 1985.

[17] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast

computing machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[18] L. Ingber, "Very fast simulated re-annealing," *Mathematical and Computer Modelling*, vol. 12, no. 8, pp. 967–973, 1989.

[19] K. Sivasundari, P. Subbaraj, and P. S. Kumar, "An effective memetic algorithm for VLSI partitioning problem," in *Proceedings of the IET-UK International Conference on Information and Communication Technology in Electrical Sciences (ICTES '07)*, pp. 667–670, December 2007.

[20] C. K. Wong and M. Sarrafzadeh, *An Introduction to VLSI Physical Design*, The Mcgraw Hill Companies, 1996.

[21] S. Venkatraman, P. Subbaraj, and P. SivaKumar, "Hybrid algorithm for Vlsi floorplanning problem," in *International Conference on VLSI and Signal Processing (ICVSP '08)*, Chennai, India, March 2008.

[22] T. Kapilachander, P. Subbaraj, and P. SivaKumar, "An effective memetic algorithm for Vlsi placement problem," in *International Conference on VLSI and Signal Processing (ICVSP '08)*, Chennai, India, March 2008.

[23] I. H. Shanavas and R. K. Gnanamurthy, "Application metaheuristic technique for solving VLSI global routing problem," in *Proceedings of the International Conference on Advances in Recent Technologies in Communication and Computing (ARTCom '09)*, pp. 915–917, October 2009.

[24] P. Subbaraj, P. SivaKumar, and Pandiraj, "Memetic algorithm for solving channel routing problems," in *International Conference on VLSI and Signal Processing (ICVSP '08)*, Chennai, India, March 2008.