Hindawi Publishing Corporation
VLSI Design
Volume 2011, Article ID 896241, 9 pages
doi:10.1155/2011/896241

Research Article

Wirelength Minimization in Partitioning and Floorplanning Using Evolutionary Algorithms

I. Hameem Shanavas[1] and Ramaswamy Kannan Gnanamurthy[2]

[1] *Department of Electronics and Communication, M. V. J. College of Engineering, Bangalore 560067, India*
[2] *Vivekanandha College of Engineering for Women, Trichengode 637205, Tamilnadu, India*

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

Received 16 December 2010; Revised 23 April 2011; Accepted 6 July 2011

Academic Editor: Zhuo Li

Minimizing the wirelength plays an important role in physical design automation of very large-scale integration (VLSI) chips. The objective of wirelength minimization can be achieved by finding an optimal solution for VLSI physical design components like partitioning and floorplanning. In VLSI circuit partitioning, the problem of obtaining a minimum delay has prime importance. In VLSI circuit floorplanning, the problem of minimizing silicon area is also a hot issue. Reducing the minimum delay in partitioning and area in floorplanning helps to minimize the wirelength. The enhancements in partitioning and floorplanning have influence on other criteria like power, cost, clock speed, and so forth. Memetic Algorithm (MA) is an Evolutionary Algorithm that includes one or more local search phases within its evolutionary cycle to obtain the minimum wirelength by reducing delay in partitioning and by reducing area in floorplanning. MA applies some sort of local search for optimization of VLSI partitioning and floorplanning. The algorithm combines a hierarchical design technique like genetic algorithm and constructive technique like Simulated Annealing for local search to solve VLSI partitioning and floorplanning problem. MA can quickly produce optimal solutions for the popular benchmark.

1. Introduction

Partitioning and floorplanning (PF) has been an active area of research for at least a quarter of a century. The main reason that partitioning has become a central and critical design task today is due to the enormous increase of system complexity in the past and the expected further advances of microelectronic system design and fabrication. Widely accepted powerful high-level synthesis tools allow the designers to automatically generate huge systems by just changing a few lines of code in a functional specification. Synthesis and simulation tools often cannot cope with the complexity of the entire system under development, and also designers want to concentrate on critical parts of a system to speed up the design cycle. Thus the present state of design technology often requires a partitioning of the system with fast and effective optimization. Moreover, fabrication technology makes increasingly smaller feature sizes and augmented die dimensions possible to allow a circuit for accommodating several million transistors. However, circuits are restricted in size and in the number of external connections. So the fabrication technology requires partitioning of a system into components by arranging the blocks without wasting free space. The direct implementation of large circuit will occupy large area. Hence the large circuit is to be split into small subcircuit. This will minimize the area of the system and the complexity of the system. When they are partitioned, the connection between two modules should be minimum (or the number net cut by the partition). This is known as cut size and hence this plays a major role in partitioning. It is a design task by applying an algorithmic method to solve difficult and complex combinatorial optimization problems like breaking a large system into pieces [1].

The process of determining block shapes and positions with area minimization objective is referred to as floorplanning. A common strategy for blocks floorplanning is to determine in the first phase and then the relative location of the blocks to each other based on connection-cost criteria.

In the second step, block sizing is performed with the goal of minimizing the overall chip area and the location of each block is finalized [2]. When the partitioning and floorplanning are combined (PF), the criteria like power, cost, and clock speed of each module are the subobjective to be optimized. The main objective to be optimized is the wirelength that can be achieved by incorporating Memetic Algorithm (MA) in both the partitioning and floorplanning [3, 4].

In early stages, many interchanging methods have been used which resulted in local optimum solution. And later some of the mathematical methods are followed. Some heuristics are also used which resulted in better result but it has its own advantage and disadvantage. Since there may be many solutions possible for this problem, stochastic optimization techniques are utilized and until now many techniques have been known like Simulated Annealing Algorithm (SA) which combines the Local Search Algorithm with the Metropolis algorithm.

SA is a simple algorithm and does not need much memory, but it takes a long time to reach the desired solution. Kernighan and Lin [5] proposed a two-way graph partitioning algorithm which has become the basis for most of the subsequent partitioning algorithms. Fiduccia and Mattheyses [6] modified the K-L algorithm to a more efficient algorithm by suggesting moving one cell at a time and by designing a new data structure separately. As a kind of global optimization technique Genetic Algorithm (GA) which borrows the concept of generation from biological system had been used for physical design problems like circuit partitioning, floorplanning, and so forth. This technique has been applied to several problems, most of which are graph related because the genetic metaphor can be most easily applied to these types of problems. GA requires more memory but it takes less time than SA [7]. Lots of researchers have proposed their theories to partition circuit using GA. Work of [8] proposed hardware genetic algorithm by developing GA and local search processor that uses some external memory to overcome the problem of local maxima/minima. Authors of [8] expressed the probability of selection of chromosome as function of both the best and worst chromosome while [4] proposed different cost functions in order to achieve multiple objectives of minimizing delay, cutsize area, and power. The authors of [9] proposed two GA one based on 0-1 encoding and other based on integer encoding. Work done in [10] developed an adaptive strategy for partitioning of circuits in which population size, crossover rate, and mutation rate are modified during the execution in order to enhance performance. Number of enhancements like crossover operator mutation or choosing different fitness functions can still be made to achieve optimal solutions. This means that theory of GA still provides opportunities for new inventions that can help in inventing new solutions for physical design problems. This paper proposes memetic algorithm to solve the graph partitioning and floorplanning problem. The algorithm incorporates several genetic algorithm features, namely, selecting a population and crossover of the selected chromosomes to get better stable solutions. The algorithm starts by incorporating the circuit as unweighted connected graph in which each vertex represents gate and edge represents an interconnection between two gates and thereafter applying the GA metaphor to generate a partition that is highly interconnected within but disconnected from other subpartitions while trying to minimize the number of cuts and time consumed. This work tried to hybrid two algorithms like Genetic Algorithm and Simulated Annealing for overcoming the disadvantage of one another. Such type of algorithms is called memetic algorithm, (see Figure 1).

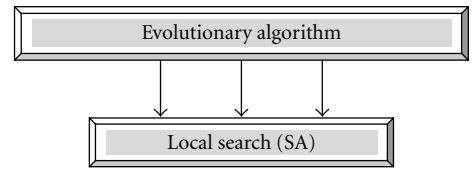

Figure 1: Memetic Algorithm.

This work addresses the problem of VLSI netlist partitioning with the objective of reducing delay and then floorplanning with the objective of reducing area to minimize the wirelength.

2. Partitioning

A better circuit partition will reduce connection among subcircuits and result in a better routing area of the layout. The challenge is that the circuit partitioning problem belongs to the class of well-known NP-hard optimization problems [11]. The problem can be viewed as a graph partitioning problem where each module (gates etc.) is taken as vertices and the connection between them represents the edges between the nodes [12, 13].

To start the algorithm, n gates are placed on the graph as n vertex, and an initial population is chosen as the different permutations of various vertices of the given graph. The problem reduces to associate each chromosome and each partition of the graph. An algorithm based on Genetic algorithm is proposed that can be used to partition a number of nodes. The example for a circuit and graph representation is given (Figure 2).

Given an unweighted connected graph $G = (V, E)$ on set of vertices V and edges E. Let $k \geq 2$ be a given integer, find a partition $V_1,\ V_2,\ V_3, \ldots, V_k$ set of vertices V such that

(i) the induced graph $G_i = (V_i, E_i)$, for all values $i = 1, 2, 3, \ldots, k$, is connected;

(ii) the following values are minimized $\min\{|V_1|, |V_2|, |V_3|, \ldots, |V_K|\}$.

This k-Connected Problem is a particular instance of graph partitioning problem [14]. Because exact and approximation algorithms that run in polynomial time do not exist for graph partitioning problems in general, it is necessary to solve the problem using heuristic algorithms. Genetic Algorithm is a heuristic technique that seeks to imitate the behavior of biological reproduction and their ability to collectively solve a problem. The GA starts with several

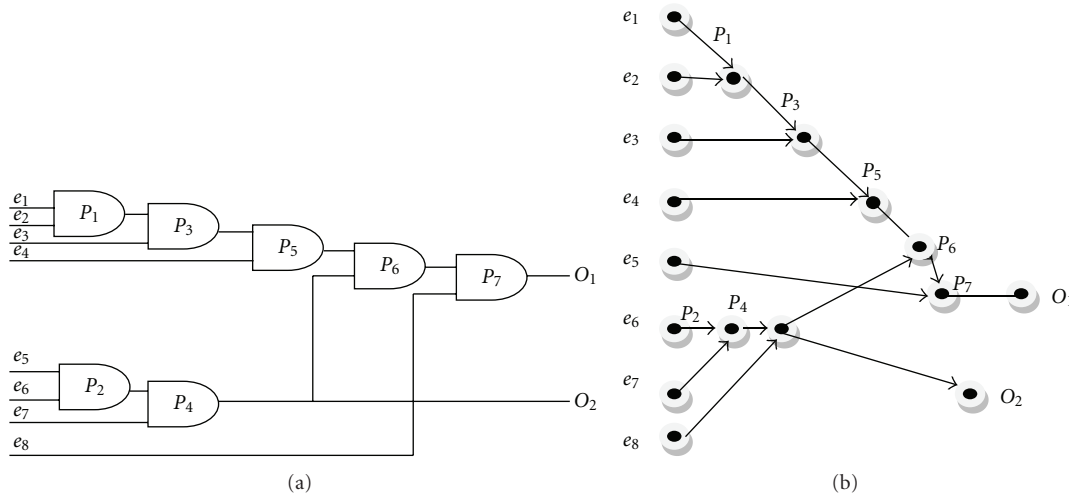

Figure 2: Representation of a circuit and Directed acyclic graph.

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 [15, 16].

3. Floorplanning

A module B is a rectangle of height H_B, width W_B, and area A_B. A supermodule consists of several modules, also called a subfloorplan, (see Figure 3). A floorplan for n modules consists of an enveloping rectangle R subdivided by horizontal lines and vertical lines into n non-overlapping 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. 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 satisfy floorplanning constraint. Given a set

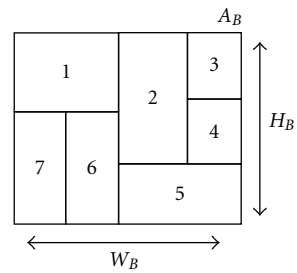

Figure 3: Floorplan module.

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 horizontal and vertical. The network graph $G = (V, E)$ has to be constructed. "$*$" represents vertical slicing of blocks. "+" represents horizontal slicing of blocks (see Figure 4).

4. Memetic Algorithm

The memetic algorithm [17, 18] is a combination of an Evolutionary Algorithm (EA) and Local Search (LS). The EAs are used for finding the global optimum. The LS used here will aid the EA to convergence speed. The evolutionary algorithm used in this work is genetic algorithm. There are two types of memetic algorithm.

Exhaustive MA. Few solutions are selected from the final generation and improved using local search.

Intermediate MA. After a predetermined number of iteration 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.

4.1. 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.

4.2. 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.

4.3. 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 the combining the two parent's genetic information. The two children produced share the characteristics of the parents as a result of this 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.

4.4. 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.

4.5. 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.

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 belongs 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 $j \cdot \mathrm{mod}2 = 1$, 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.

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. The sum of the weighting factors equals one. The complete fitness function is

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

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

At the beginning, a set of Polish expressions is given for floorplannig. 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.

$$\text{Total Area} = \sum A_n. \tag{2}$$

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 is

$$G_f = \left\{ \frac{A_a}{A_m} > 1, \frac{A_a}{A_m} \leq 1, \frac{A_a}{A_m} * A_f \right\}. \tag{3}$$

5. Local Search

After a prescribed number of iterations by evolutionary algorithm, local search algorithm is applied to few random individual, to have better solution. Local search methods are iterative algorithms that tend to enhance solution by stepwise improvement and make an attempt to reach optimum solutions, which 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 again 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 crossaover, it can also be used for parents selection, and used before or after mutation to increase the number of fitness variables (see Algorithm 1).

5.1. 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 as 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 [19] and Cerny in 1984 [20]. 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 decrease the temperature. At thermal 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. [21]. 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

$$p = \frac{p_r}{p_q} = \frac{\exp\left(-E_t - E_q\right)}{K_t} > 1. \tag{4}$$

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 [22].

```
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 ε 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

5.2. 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 are given in (4). 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 wire length. A formal description of the simulated annealing algorithm was given in Section 5. There are two methods for generating new configurations from the current configuration. 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.

$$\begin{aligned} L_{WV}(T) &= L_{WV}(T_i)\left[\frac{\log T}{\log T_i}\right], \\ L_{WH}(T) &= L_{WH}(T_i)\left[\frac{\log T}{\log T_i}\right], \end{aligned} \tag{5}$$

where T is the current temperature, T_i is the initial temperature, $L_{WV}(T_i)$ and $L_{WH}(T_i)$ are the initial values of the vertical and horizontal window spans $L_{WV}(T)$ and $L_{WH}(T)$, 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:

$$C = \sum_{\text{nets } i} [x(i)W_H(i) + y(i)W_V(i)], \tag{6}$$

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., $E_t - E_q$) is the cost increase and T is the current temperature. When the cost increases, or when the temperature decreases, the acceptance probability (4) 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 $T < 0.1$.

5.3. Floorplannning Based on Simulated Annealing. This section describes an optimal floorplanning on simulated annealing algorithm. Assume that a set of modules are 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. 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 trees in Figure 4 represent the given floorplan. 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.

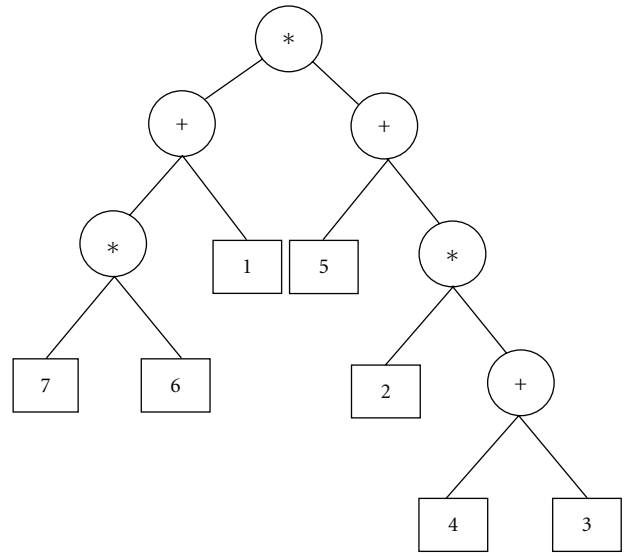

Figure 4: Graph representation of the given floorplan module. *Polish expression:* 76∗1+43+2∗5+∗.

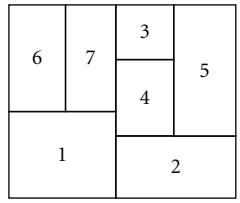

Figure 5: Condition 1 (67∗1+34+2∗5+∗).

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

Condition 2. Complement a series of operators between two operands (67+1∗34∗2+5∗+) (see Figure 6).

Condition 3. Exchange adjacent operand and operator if the resulting expression is a normalized Polish expression (67+43∗+25∗1+∗) (see Figure 7).

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

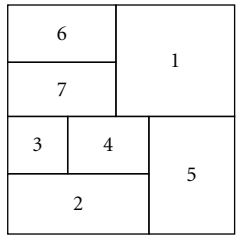

Figure 6: Condition 2 (67+1∗34∗2+5∗+).

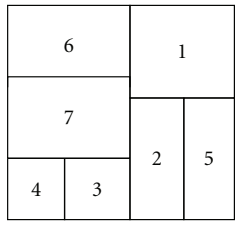

Figure 7: Condition 3 (67+43∗+25∗1+∗).

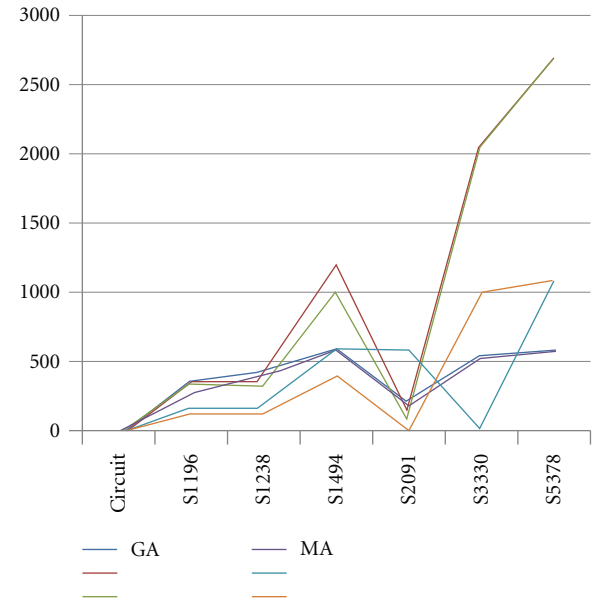

Figure 8: Comparison of MA with GA.

Table 1: Delay comparison of MA with GA for ISCAS89 benchmark.

Circuit	GA			MA		
	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	47	994
S5378	587	2687	2686	590	1078	1100

Polish expression. There are two components for the cost function, area, and wire length. The area of a sliceable floorplan can be computed easily using the floorplan sizing algorithm. The wire-length 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 5, 6, and 7 shows a series of movements which lead to a solution.

6. Experimental Results

The memetic algorithm has been implemented in C++ on a Linux installed personal computer. In this work, it compares the performance of a pure genetic algorithm to a memetic technique that combines GA with simple local search techniques like Simulated Annealing. In circuit partitioning, MA can increase the speed of execution time on an average of 45% when compared to simple genetic algorithm see Table 1 and Figure 8. 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 wirelength minimization can be achieved by reducing the delay in circuit partitioning. In Floorplanning, wirelength and CPU time are compared with simulated annealing see Table 2. MA can reduce the wirelength on an average of 0.5 mm when compared with Fast Simulated annealing see (Figures 9 and 10).

Table 2: Wirelength comparison of MCNC benchmark for n100, n200, & n300 WITH ASPECT RATIO $R = 1, 2, 3, 4$.

Circuit	Aspect ratio R*	Fast-SA		MA	
		Wire (mm)	Time (sec)	Wire (mm)	Time (sec)
n100	1	33.56	30	32.06	26
	2	35.44	30	34.39	24
	3	35.48	30	34.23	27
	4	36.89	29	32.74	27
n200	1	63.55	175	58.33	150
	2	62.76	173	59.84	156
	3	63.70	180	61.55	156
	4	66.31	176	63.72	171
n300	1	76.05	399	71.00	363
	2	77.60	386	74.22	358
	3	81.67	391	79.56	371
	4	88.58	382	82.18	370
Comparison		1.06	1.11	1.00	1.0

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 45%

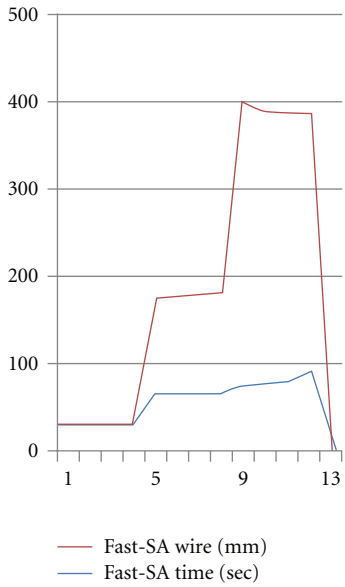

Figure 9: Performance of Fast Simulated Annealing.

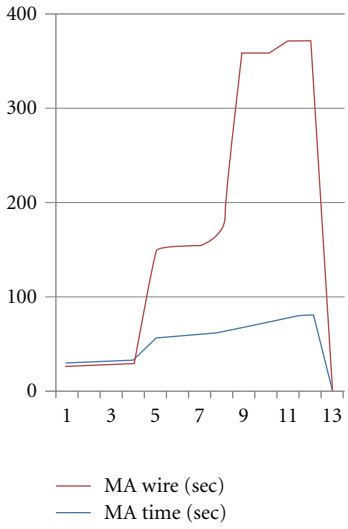

Figure 10: Performance of Memetic Algorithm.

faster than the simple software genetic algorithm by reducing the delay and area in partitioning and floorplanning, respectively, that would indefinitely reduce the wirelength. The implementation of multiobjective in the algorithm enables to get the near optimal solution. After a predetermined number of iteration by GA, local search is applied to few random individual to get the optimal solution by Simulated Annealing. In the future, the performance of the algorithm has to be tested on different benchmarks.

References

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

[2] I. Hameem Shanavas, R. K. Gnanamurthy, and T. Stephen Thangaraj, "Hybrid algorithmical approach for VLSI physical design—floorplanning," *International Journal of Recent Trends in Engineering*, vol. 2, no. 4, 2009.

[3] M. Tang and X. Yao, "A memetic algorithm for VLSI floor planning," *IEEE Transactions on Systems, Man, and Cyber Net*, vol. 37, no. 1, 2007.

[4] P. Subbaraj, K. Sivasundari, and P. Siva Kumar, "An effective memetic algorithm for VLSI partitioning problem," in *Proceedings of the International Conference on ICTES*, pp. 667–670, Chennai, India, 2007.

[5] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell System Technical Journal*, pp. 291–307, 1970.

[6] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proceedings of the 19th ACM IEEE Design Automation Conference*, pp. 175–181, 1982.

[7] T. W. Manikas and J. T. 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.

[8] Y. Yodtean and P. Chantngarm, "Hybrid algorithm for circuit partitioning," in *Proceedings of IEEE Region 10 Conference*, vol. 4, November 2004.

[9] G.-F. Nan and M.-Q. Li, "Two novel encoding strategies based genetic algorithms for circuit partitioning," in *Proceedings of the 3rd IEEE International Conference on Machine Learning*, vol. 4, pp. 2182–2188, Shangai, 2005.

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

[11] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to Theory of NP-Completeness*, Freeman, San Francisco, Calif, USA, 1979.

[12] G. Tumbush and D. Bhatia, "Partitioning under timing and area constraints," in *Proceedings of the International Conference on Computer Design*, pp. 614–620, October 1997.

[13] 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*, pp. V-521–V-524, May 2004.

[14] S. M. Sait, A. H. El-Maleh, and R. H. Al-Abaji, "General iterative heuristics for VLSI multiobjective partitioning," in *Proceedings of the 2003 IEEE International Symposium on Circuits and Systems (ISCAS '03)*, pp. V497–V500, May 2003.

[15] F. M. Johannes, "Partitioning of VLSI circuits and systems," in *Proceedings of the 1996 33rd Annual Design Automation Conference*, pp. 83–87, June 1996.

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

[17] 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, 2009.

[18] N. Krasnogor and J. Smith, "A tutorial for competent memetic algorithms: model, taxonomy, and design issues," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 5, pp. 474–488, 2005.

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

[20] V. Cerny, "Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm," *Journal of Optimization Theory and Applications*, vol. 45, no. 1, pp. 41–51, 1985.

[21] 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.

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