IJST (2015) 39A3 (Special issue): 375-382
Iranian Journal of Science & Technology

http://ijsts.shirazu.ac.ir

A novel differential evolution based optimization algorithm for Non-Sliceable VLSI floorplanning

D. Gracia Nirmala Rani* and S. Rajaram

Department of Electronics and Communication Engineering, Thiagarajar College of Engineering, Madurai, 625015, Tamilnadu, India
E-mail: gracia@tce.edu

Abstract

Floorplanning is an important step in physical design of VLSI circuits. It is used to plan the positions of a set of circuit modules on a chip in order to optimize the circuit performance. However, modern floorplanning takes better care of providing extra options to place dedicated modules in the hierarchical designs to align circuit blocks one by one within certain bounding box for helping sequential data transfer (bus or pipeline) signal in the VLSI circuit. In this paper, the placement of circuit blocks with alignment constraints can be handled using B*tree representation with Differential Evolutionary algorithm. In order to reduce the solution space, feasibility conditions of nonslicing floorplan with alignment constraints have been examined. The properties associated with our proposed Differential Evolutionary algorithm provide the way to produce optimal floorplan. Experimental results based on the MCNC benchmark circuits with the alignment constraint shows that our Differential algorithm can produce promising solutions.

***Keywords**:* CAD VLSI; floorplanning; alignment constraints; differential evolutionary algorithm

1. Introduction

Floorplanning problem plays a significant role in physical design of VLSI circuits and the problem is shown to be NP-complete (Stockmeyer L, 1983). Classical floorplanning formulation can be used to determine the location of given set of modules without overlapping each other. The main objective of the floorplanning is to minimize area and wire length (Chang-Tzu Lin et al, 2002; Valenzuela C.L et al, 2002). In this floorplanning step, the VLSI designer will pay more attention to giving extra options to place modules in the final packing for various reasons. If there are many interconnections between the modules, and to provide a space for bus based routing, the VLSI circuit designer has to be careful regarding the separation between two modules. Therefore, it is desirable to find an efficient and effective way to handle the floorplanning problem with the placement constraints.

Floorplanning structure can be divided into two types, one is sliceable and another one is non-sliceable. For slicing structure, Otten (1982) first introduced a binary tree representation, and later Wong and Liu (1986) proposed a normalized Polish expression. Young et al (2001) proposed an algorithm to handle abutment constraint based on slicing structure. Chang et al (2005) also presented the floorplanning with abutment and fixed outline constraints. Though sliceable structure has so many advantages, typical floorplan is always non-sliceable.

There are several representations for non-slicing structures such as BSG (S. Nakatake et al, 1996), sequence pair (H. Murata et al, 1995), O-tree (P.N. Guo et al, 1999), B*-tree (Y. C. Chang et al, 2000), CBL (Hong et al, 2000), and Q-sequence (K. Sakanushi and Y. Kajitani, 2000) right, below, and above. These representations are used for general floorplanning problem. Block placement can be obtained by compacting blocks subject to relative positions. O-tree and B*-tree are used to describe one dimensional relative positions among blocks. The final placement is achieved by compacting blocks to the left and bottom. Even though O-tree and B*-tree have smaller solution space, they are dimension dependent. Because of the geometric relation between blocks, solution cannot be obtained directly from representation.

Ma et al (2001) proposed a novel algorithm for Non-slicing representation with abutment constraints. In case of constraints violation, simulated annealing process is used to examine the intermediate solutions and repair the solutions by heuristics method (penalty function). Adya et al

*Corresponding author
Received: 25 July 2013 / Accepted: 21 January 2015

(2001) suggested new objective functions to drive simulated annealing and types of move to help exploit fixed-outline floorplans. The new move is helpful in reducing critical path. Meng-Chen Wu and Yao-Wen Chang (2004) proposed the placement problem with alignment constraints using B*tree representation. Still, it does not guarantee an optimal floorplan with fixed-outline constraint being produced. Rong Liu et al (2005) recommended the sequence pair representation for nonslicing Floorplanning with boundary and pre-placement constraints. VLSI Block placement using alignment constraints is proposed by Song Chen et al (2006). Then, Wan-Ping Lee et al (2009) developed a dynamic-programming based voltage scaling algorithm and a timing driven Floorplanning with the B*tree representation using multi supply voltage (MSV) design.

Po-Hsun Wu and Tsung-Yi Ho (2012) developed a Bus driven VLSI Floorplanning with thermal consideration. They used the sequence pair representation with simulated annealing algorithm for effectively separate hotspots and reduce the chip temperature in the VLSI Floorplanning.

In this paper, we present the modern floorplanning with the alignment constraints using the nonslicing representation. First, we consider the feasible properties of a non-slicing floorplan with alignment constraints. These properties, together with our proposed Differential evolutionary algorithm provide the way to produce optimal floorplan.

2. Preliminaries

The non-slicing floorplans are handled using an ordered binary-tree representation called B*tree representation. We can construct a B*tree for a given admissible placement. [See Fig. 1(b) for the B*tree representing and the placement is shown in Fig. 1(a)]. A B*tree is an ordered binary tree. Each node n_i in a B*-Tree denotes a module. The root of a B*-Tree corresponds to the module on the bottom-left corner. We have to construct the B*tree for an admissible placement P in a recursive fashion which is similar to the depth first search procedure.

Starting from the rooting node, we build the left sub tree and then the right sub tree in recursively manner. The geometric relationship between two modules in a B*tree will be explained as follows. Each node n_i in a B*-Tree denotes a module. The root of a B*-Tree corresponds to the module on the bottom-left corner. The left child n_j of a node n_i denotes the module b_j that is the lowest adjacent module on the right-hand side of b_i i.e. ($x_j = x_i + w_i$). The right child n_k of a node n_i denotes the module b_k that is the lowest visible module above b_i and with the same x co-ordinate as b_i i.e. ($x_k = x_i$).

Figures 1(a) and 1(b) show a placement and its corresponding B*-Tree respectively. The root n_0 of the B*-Tree in Fig. 1(b) denotes that b_0 is the module on the bottom-left corner of the placement. For node n_3 in the B*tree, n_3 has a left child n_4 which means that module b_4 is the lowest adjacent module in the right-hand side of module b_3 (i.e. $x_4 = x_3 + w_3$). n_7 is the right child of n_3 since module b_7 is the visible module over module b_3 and the two modules have the same x co-ordinate ($x_7 = x_3$).

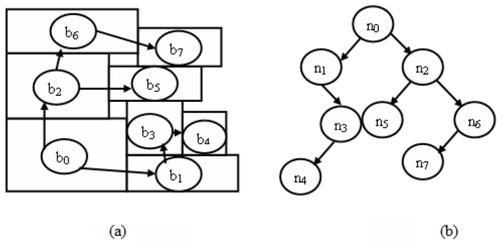

Fig. 1. (a) An admissible placement. (b) The B*tree representing the placement

3. Alignment Constraints

Nowadays, VLSI circuit designers are focused the alignment constraints for facilitating sequential data transfer operation. As it is different from rectilinear blocks with fixed relative positions, the alignment blocks are flexible to move within the pre-specified alignment range. Similar to a bus width, the alignment blocks must be placed one by one horizontally or vertically in a predefined range. Therefore, the alignment can be divided into H-alignment and V-alignment according to the direction of blocks either horizontal or vertical. The horizontal and vertical alignment diagrams are shown in Fig. 2.

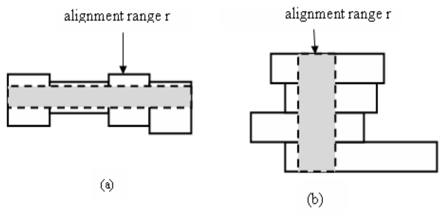

Fig. 2. (a) Blocks with H-alignment (b) Blocks with V-alignment

4. Floorplanning with Alignment Constraints

4.1. Problem Definition

The floorplanning problem can be stated as follows: Consider B = {b_1, b_2…b_n} will be a set of n rectangular blocks. If we let w_i be the width of module i and h_i be the height of module i, then the area of the module i will be calculated as A = w_i x

h_i, 1≤ i≤ n. The bottom-left corner of block b_i will be coordinated as (x_i, y_i). No two blocks can overlap and the given alignment constraints should be satisfied when we place the blocks in the chip. The main objective of floorplanning/placement is to optimize a predefined cost metric such as area (i.e., the minimum rectangle of P) and wire length (i.e., the sum of all interconnection lengths) of the chip.

*4.2. B*Tree with Alignment blocks*

For a B*-tree, the left child n_j of the node n_i represents the lowest adjacent block b_j which is right to block b_i (i.e. $x_j = x_i \mid w_i$). If the corresponding nodes of blocks form a left skewed sub tree, the blocks can be adjacent to one by one in a B*tree representation. There are four sets of abutment blocks b_0 and b_1, b_3 and b_4, b_2 and b_5, and b_6 and b_7 correspond to four left-skewed sub-trees in the given example Fig. 5.

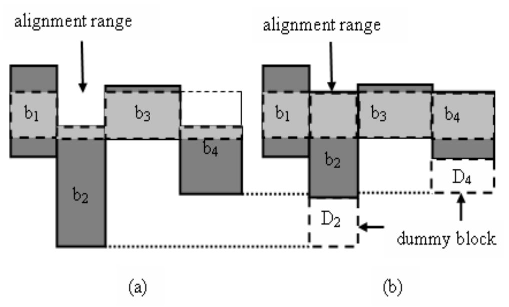

Fig. 3. (a) An infeasible placement with blocks falling out of alignment range; block b2 and b4 are not in the alignment range. (b) Insertion dummy blocks, we obtain a feasible placement without any blocks violating the alignment constraints

After packing, the blocks are arranged in the bottom left corner. The blocks which are joined with a left-skew sub-tree of a B*-tree may be aligned together if no blocks fall during packing. To solve the problem of block falling, *dummy* blocks are introduced to fix it. As the dummy blocks can be arranged near to the alignment range r, the dummy block will have the same x-coordinate with the alignment block and right below it. Attempting to set width of the dummy block equal to its corresponding alignment block. This dummy block is simply left in corresponding alignment block, and the height can be adjusted to make a displaced alignment block allowing shifted in to right alignment range. As illustrated in Fig. 3(a), the blocks b_2 and b_4 will fall out of the alignment range. According to the alignment range, the dimensions are adjusted in Fig. 3(b). We have adjusted the height of the two dummy blocks to shift the displaced alignment blocks. After adjusting the heights of the dummy blocks, we make sure that the resulting placement height recommendations will be possible with the alignment constraints.

4.3. Feasibility condition for Alignment constraints

Given a B*-tree construction, the node representing an alignment block should be considered as an alignment node. On the alignment node, it is introduced in the dummy node within the B*-tree, so that the alignment node is connecting the right child of its corresponding dummy node. According to the definition of the B*-tree, the dummy block is to be placed right under its corresponding alignment block. Then we modify the height of the dummy block to change the y-coordinate of the alignment block, if needed.

The combination of alignment node and its corresponding dummy node will be called as a cluster node. For example, three cluster nodes n_3, n_4 and n_5 will make a left-skewed sub-tree in an alignment shape (Fig. 4(a)) whose placement for the alignment blocks b_3, b_4, and b_5 will be one by one as shown in Fig. 4(b). To develop the feasibility condition of a B*tree with the alignment constraints, we will follow the theorem (Meng-Chen Wu and Yao-Wen Chang, 2004) given below,

Theorem: If the alignment nodes in a B*tree form an alignment shape, there is a feasible placement with alignment constraints will exist.

When dealing with the alignment constraints, two passes are required to pack blocks correctly. In the first operation, the coordinate of each non-dummy blocks (a regular block or an alignment block) should be calculated. Then, it can be checked whether any alignment block is out of range or not. If there is any violation, it should be computed in the second pass which is the minimum movement (height) for the corresponding alignment (dummy) block to shift into the alignment range.

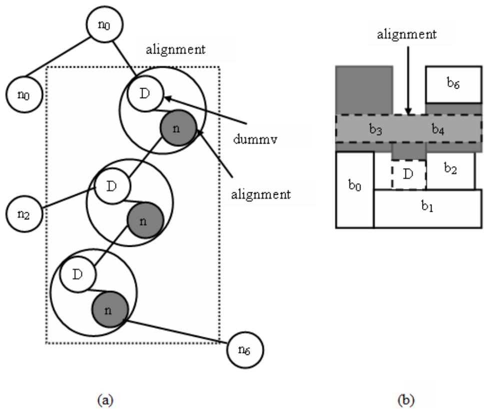

Fig. 4. (a) The alignment shape in a B*tree. (b) The corresponding placement of (a)

If k alignment blocks and the alignment range r are given, the equation for computing the minimum movement (height) Δ_i for the alignment block $b_{i,}$ i=1, 2….k in H-alignment is as follows:

$$\Delta_i = \begin{cases} y_{\max} + 1 \le y_i + h_i & \text{if } 1 \le i \le k \\ 0 & \text{otherwise} \end{cases}$$

where

$$y_{\max} = \max \{ y_i \mid i=1, 2, 3 \ldots k\} \quad (1)$$

After obtaining this minimum movement for each alignment block, we have to set the height of the corresponding dummy block to Δ_i to move the alignment block towards the alignment range. We can ensure that the final placement will be feasible without violating any alignment constraint by using such a two-pass packing scheme. The alignment blocks b_2, b_3, b_4, and $b_{5,}$ one by one, but the blocks b_2 and b_5 will fall out of the alignment range after the 1st-pass packing as shown in Fig. 5(a). Then, we must compute the minimum movement (Δ_i) for each dummy blocks in the alignment shape and b_2 and b_5 blocks should be shifted to upward by Δ_2 and δ_5 respectively. Figure 5(b) produce a good placement after the adjustment.

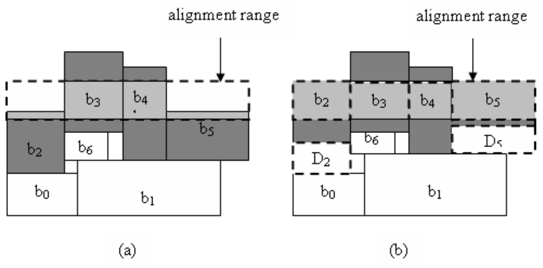

Fig. 5. (a) The placement obtained in 1st pass, where blocks 2 and 5 fall out of the alignment range. (b) The placement obtained in the 2nd pass, where those blocks violating the alignment constraint are adjusted by inserting corresponding dummy blocks with appropriate heights

5. Differential Evolutionary Algorithm

5.1. Fitness Function

In the Differential Evolution (DE), each individual in the population is an admissible floorplan represented by a B*-tree. The main objective of the VLSI floorplan is to minimize the cost of floorplan such as area and wire length, the fitness of an individual in the population is formulated as follows:

$$f(F) = \frac{1}{cost(F)} \quad (2)$$

Where F is the corresponding floorplan, and cost (F) is the cost of F defined as,

$$\text{cost}(F) = \alpha \times \frac{area}{area^*} + \beta \times \frac{wirelength}{wirelength^*} \quad (3)$$

5.2. Initial Population

An admissible VLSI floorplan F is represented by individual in the initial population in B*tree. Admissible B*tree is designed by using constructive algorithm. The constructive algorithm is based on a deterministic algorithm proposed by Chang et al (2000).

For each rectangular module B, we have to calculate the function V by using this algorithm. According to the value of V, we can include the modules into an initially empty B*tree B. Then we have to check a point such that the new module will be packed in the feasible region in the bottom left corner. The constructive algorithm is invoked iteratively to create an initial population of individuals.

5.3. Differential Evolutionary Operators

In this technique, the Differential Evolutionary algorithm (Price, et al, 1995) is a population based algorithm derived from genetic algorithms by using similar operators like crossover, mutation and selection. The implementation of operators depends on constructing better solutions in genetic algorithms that rely on crossover at the same time Differential Evolution relies on mutation operation. The main operation is based on the divergence between random sample pairs of solutions in the population.

The algorithm is a search mechanism used as mutation operation inspired by selection operation to direct the search toward the prospective regions in the search space. It is also used as a non-uniform crossover for finding operations like child vector parameters from one parent more often than it does from others. The solutions obtained were better using the various components like trial vectors; the recombination (crossover) operator efficiently shuffles information about successful combinations, enabling the search for a better solution space.

An optimization task consisting of D parameters can be represented by a Dimensional vector (Price, et al, 1995). In Differential Evolution, a population of NP solution vectors is randomly created at the start. This population is successfully improved by applying mutation, crossover and selection operators.

5.3.1. Mutation

A mutant vector is generated according to the following formula for each target vector $x_{i,G}$,

$$V_{i,G-1} = X_{r1,G} + F.(X_{r2,G} - X_{r3,G}) \quad (4)$$

Randomly chosen indexes r_1, r_2, r_3 ε {1, 2, 3…NP}. It must be noticed that indexes have to be different from each other and also from the running index. Hence, the number of parameter vectors in a population must be at least four, which is a real and constant factor ε [0, 2], F is the scaling vector that controls the amplification of the difference vector ($x_{r2,G}$ - $x_{r3,G}$). The smaller difference between the parameters of parent r_2 and r_3 shows that the step length is automatically decreased if the population gets close to the optimum. Two sub trees with the same tree height at a level were selected randomly. Then, these two sub-trees will swap with each other which is equal to swapping two modules. It is one of the basic operations for local search on the B*tree.

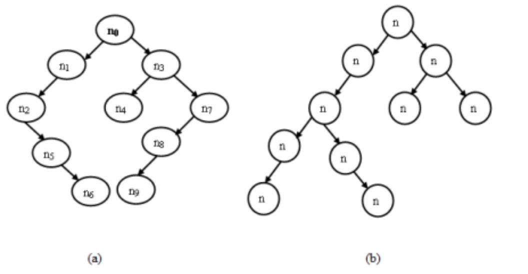

Fig. 6. Mutation operators

5.3.2. Crossover

The target vector is mixed with the mutated vector to use the following scheme to yield the trial vector

$$u_{i,G}-1 = \left(u_{1i,G+1}, u_{2i,G+1} \ldots\ldots\ldots u_{Di,G+1}\right) \quad (5)$$

Where

$$U_{ij,G-1} = \begin{cases} V_{i,G-1}\ if(r(j) \le CR \text{ or } j = m(i) \\ X_{ji,G}\ if(r(j) > CR \text{ and } j \ne m(i) \end{cases}$$

If any two individuals are considered as alleged parents, both of them are admissible floorplans represented by B*tree. The crossover operator can exchange important structural information from two parents to a child. Firstly, a partial set of nodes move to the offspring through this process straight from the first parent. Hence, the child can have some inherited properties from their parents. In the second case, the remaining nodes are selected from the second parent in an arbitrary direction.

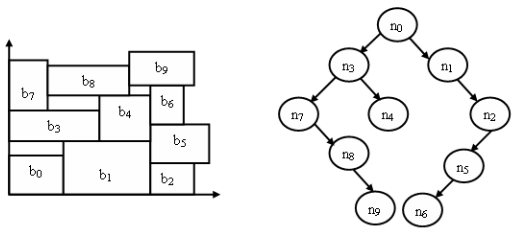

Fig. 7. P1 Parent

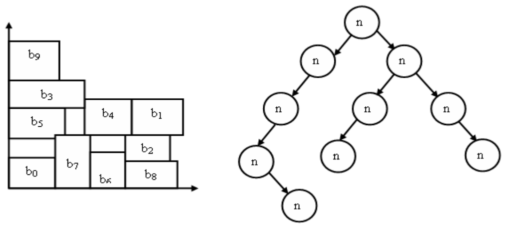

Fig. 8. P2 Parent

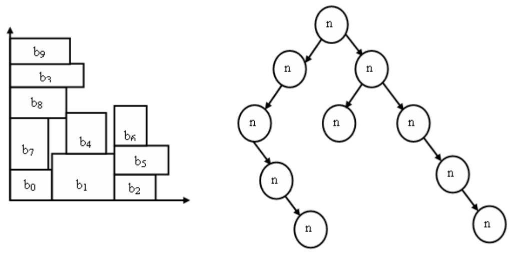

Fig. 9. C1 Child

The child C_1 (Fig. 9) produced this way has structural information from two parents P_1 (Fig. 7) and P_2 (Fig. 8). In this paper, the crossover operator is selected and creates the duplicates copy of structural information from the left branch from P_1 and puts it in Cl. Then, the crossover operator takes a copy from P_2 and removes those nodes that have already been present in C_1 and then adds the remaining structural components to C_1. In this way, after generating a child it carries the significant structural information from the parents.

5.3.3. Selection

The trail vector is accepted as a new parent vector for the following G+1, if only it gives a better cost function value when compared to the parameter vector. It is accepted as a new parent vector for the following generation G+1. Otherwise, the target vector is retained to serve as a parent vector for generating G+1 once again.

The Pseudo code for our proposed algorithm is given below.

Algorithm: Placement with Alignment Constraints (blocks, constraints)

```
Input: A set of blocks with alignment constraints.
Output: An optimal placement without violating the given constraint.
1. Initialize a B*-tree for the input blocks and constraints;
2. Differential Evolutionary Process;
3. Begin
4. Initialize the parameters
5. Evaluate the cost of the vector
6. Mutation
7. Crossover
8. Selection
9. Evaluate the B*-tree cost.
10. until criteria are met.
11. End
12. Return the best solution;
```

6. Experimental results

The experiment employed MCNC benchmarks for the VLSI floorplanning with alignment constraints. We have compared our method to Meng-Chen Wu and Yao-Wen Chang (2004), the most recent Differential Evolution based performance driven VLSI Floorplanning. For a fair comparison, we adopt the same cases used in Meng-Chen Wu and Yao-Wen Chang (2004), as shown in Table 1. All experiments were run on the same machine. Our approach adopts a Differential Evolutionary algorithm without resorting to floorplan representations, and the placement constraints of the buses are satisfied during the block-packing process.

For the Differential Evolutionary algorithm, crossover (CR) and Scaling factor (F) are the control parameters. Like Number of Population (*NP)*, both values remain constant during the search process. *F* is a real-valued factor in the range (0.0, 1.0) that scales the differential variations, and therefore controls mutation amplifications. Respectively, *CR* is a real-valued crossover factor in range [0.0, 1.0] that controls the probability that a trial vector parameter will come from the randomly chosen, mutated vector, $V_{j,i,G+1}$, instead of from the current vector, $X_{j,i,G}$. Usually, suitable values for *F*, *CR* and *NP* can be found by trial-and-error after a few tests using different values. In our experiment, we have obtained better optimal results by setting NP to 100, the probability for crossover (CR) is 0.9 and Scaling Factor F=0.5 compared with the Simulated Annealing. The comparison between the various optimization algorithms without constraints is shown in Table 1. The area and run time comparison with constraints results are shown in Table 2 and Table 3.

Table 1. Area Estimation without Constraints

MCNC Benchmarks		Non Slicing floorplanning without Constraints					
		Previous research work					Proposed method
Circuits	**Module number#**	**Simulated Annealing Area (mm^2)**	**SA embedded inTabu searchArea(mm^2)**	**Evolutionary Simulated Annealing mm^2)**	**Hybrid Simulated annealing (mm^2)**	**Hybrid Genetic (mm^2)**	**Differential Evolution Area(mm^2)**
apte	9	46.92	46.92	49.43	47.12	46.90	46.6339
Xerox	10	19.83	20.08	20.78	20.89	20.03	19.69
Hp	11	8.95	9.16	9.364	9.47	9.08	9.293
Ami33	33	1.27	1.21	1.253	1.21	1.19	1.22
Ami49	49	36.80	36.95	36.74	37.80	37.49	36.22

Table 2. Area Estimation with Constraints

Circuit	Blocks	Constrained Blocks Alignment	Meng and Chang Area (mm^2)	Ours Area (mm^2)
Apte	9	4	46.92	45.339
xerox-1	10	4	20.08	19.17
xerox-2	10	4	20.08	19.98
hp-1	11	4	9.20	8.85
hp-2	11	4	9.349	9.00
ami 33-1	33	4	1.180	1.17
ami 33-2	33	4	1.181	1.22
ami 49-1	49	5	36.60	35.36
ami 49-2	49	4	36.56	35.55
ami 49-3	49	4	36.64	35.60

Table 3. Run Time Estimation with Constraints

Circuit	Blocks	Constrained Blocks Alignment	Meng and chang Time(sec)	Ours Time (sec)
Apte	9	4	3.6	1.48
xerox-1	10	4	5.8	4.09
xerox-2	10	4	6.4	5.12
hp-1	11	4	5.9	3.98
hp-2	11	4	6.1	4.04
ami 33-1	33	4	35.4	20.98
ami 33-2	33	4	52.6	25.78
ami 49-1	49	5	132.7	87.98
ami 49-2	49	4	97.9	78.56
ami 49-3	49	4	109.2	80.09

7. Conclusion

In this paper, we have proposed an efficient and effective Differential Evolutionary algorithm for floorplanning based on B*tree representation. The proposed DE (Differential Evolution) algorithm guarantees a feasible and good placement solution with alignment constraints for giving support to sequential data transfer (bus or pipeline signals). MCNC benchmark circuit's data are used for testing and the results are very promising. Our approach generates a good placement solution with an optimum area and a short run time. Another major advantage using DE is the reduction of computation time by proposing a trial solution (with all constraints) which will not require evaluating the objective function every time. Also we have obtained good packings with the alignment constraints satisfied efficiently. The future scope of this work is to include boundary constraints, range constraints, rectilinear constraints and performance constraints to the proposed algorithm.

References

Chang, Y. C., & Chang, Y. W, Wu, G. M., & Wu, S. W. (2000). B*tree: A New representations for nonslicing floorplans. *Proc. ACM/IEEE Design Automation Conf.*, Los Angles, 458–463.

Chang-Tzu Lin, De-Sheng Chen, Yi-Wen Wang, & Hsin-Hsien Ho. (2005). Modern Floorplanning with Abutment and Fixed-Outline Constraints. *IEEE International Symposium on Circuits and Systems*, 6, 6214–6217.

Chang-Tzu Lin, De-Sheng Chen, & Yiwen.Wang. (2002). An Efficient Genetic Algorithm for Slicing Floorplan Area Optimization. *Proceedings of the IEEE International Symposium on Circuits and Systems*, 2, 879–882.

Guo, P. N., Cheng, C. K., & Yoshimura, T. (1999). An O-tree representation of non- slicing floorplans and its applications. *Proc. DAC*, 268–273.

Hong, X. L., Huang, G., & Cai, Y. C. et al. (2000). *Corner block list*: An effective and efficient topological representation of non-slicing floorplan. Proceedings of IEEE/ACM International Conference on Computer-Aided Design, 8–12.

Murata, H., Fujiyoshi, K., Nakatake, S., & Kajitani, K. (1995). Rectangle-packing based module placement. *Proceedings of the IEEE/ACM International conference on Computer-aided design*, 472–479.

Nakatake, S., Murata, H. Fujiyoshi, K., & Kajitani Y. (1996). VLSI module placement on BSG-structure and IC layour applications. *Proceedings of the IEEE/ACM International conference on Computer-aided design*, 484–491.

Otten, R. H. J. M. (1982). Automatic floorplan design. *Proc. DAC*, 61–267.

Sakanushi, K., & Kajitani, Y. (2000). The quarter-state sequence (Q-sequence) to represent the floorplan and applications to layout optimization. *Proc. APCAS*, 829–832.

Stockmeyer, L. (1983). Optimal orientations of cells in slicing floorplan designs. *Information and Control*, 59, 91–101.

Valenzuela, C. L., & Wang, P. Y. (2002). VLSI placement and area optimization using a genetic algorithm to breed normalized postfix expressions. *IEEE Transactions on Evolutionary Computation*, 6(4), 390–401.

Wong, D. F., & Liu, C. L. (1986). A new Algorithm for floorplan design, *Proc. DAC*, 101–107.

Young, F. Y., Yang, H. H., & Wong, D. F. (2001). On extending slicing floorplans to handle L/T-shaped blocks and abutment constraints, *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, 20(6), 800–807.

Yuchun Ma, Xianlong Hong, Sheqin Dong, Yici Cai, Chung-Kuan Cheng, & Jun Gu. (2001). Floorplanning with abutment constraints based on corner block list. *Integration VLSI Journal*, 65–77.

Saurabh, N. A., & Markov, L. I. (2001). Fixed-outline FloorplanningThrough Better Local Search. *Proceedings of the International Conference on Computer Design*, 328–334.

Meng-Chen Wu, & Yao-Wen Chang. (2004). Placement with Alignment and Performance Constraints Using the B*-tree Representation. *Proceedings of the IEEE International Conference on VLSI in computers and Processors*, 568–571.

Rong Liu, Sheqin Dong, Xianlong Hong, & Yoji Kajitani. (2005). Fixed-outline Floorplanning with Constraints through Instance Augmentation. *IEEE International symposium on circuits and systems*, 2, 1883–1886.

Song Chen, Sheqin Dong, Xianlong Hong, Yuchun Ma,

& Cheng, C. K. (2006). VLSI Block Placement With Alignment Constraint. *IEEE Transactions on Circuits and Systems: Express Briefs*, 53(8).

Wan-Ping Lee, Hung-Yi Liu, & Yao-Wen Chang. (2009). Voltage-Island Partitioning and Floorplanning under Timing Constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(5).

Po-Hsun Wu, & Tsung-Yi Ho. (2012). Bus-driven floorplanning with thermal consideration. *Intergration, the VLSI journal*, http://dx.doi.org/10.1016/j.vlsi.2012.11.002.

Stron, R., & Price, K. (1995). Differential evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces.Tech.Rep.TR-95–102, *International Computer Science Institute (ICSI)*, Berkeley, California, USA.