Hybrid Genetic Algorithm for VLSI Macro Cell Layout

Sathiamoorthy S. and Andaljayalakshmi G.

Department of Computer Science and Engineering,
Thiagarajar College of Engineering, Madurai -625015. INDIA

Abstract. Genetic algorithms have proven to be a well-suited technique for solving selected combinatorial optimization problems. The blindness of the algorithm during the search in the space of encoding must be abandoned, because this space is discrete and the search has to reach feasible points after the application of the genetic operators. This can be achieved by the use of a problem specific genotype encoding, and hybrid, knowledge based techniques, which support the algorithm during the creation of the initial individuals and the following optimization process. In this paper a novel hybrid genetic algorithm, which is used to solve macro-cell placement problem is presented. Two new heuristics are introduced. Due to a tree-structured genotype representation and hybrid, problem- specific operators, the proposed approach is able to show satisfactory performance.

1   Introduction

The design of VLSI *(very large scale integrated)* microchips is a process of many consecutive steps including specification, functional design, circuit design, physical design, and fabrication. Macro-cell layout [2] generation is a task in the *physical design cycl*e. The circuit is partitioned and the components are grouped in functional units, the *macro-cell*s. These cells can be described as rectangular blocks with *terminals* (pins) along their borders. These terminals have to be connected by *signal net*s, along which power or signals (e. g., clock ticks) are transmitted between the various units of the chip. A net can connect two or more terminals, and some nets must be routed to *pads* at the outer border of the layout, since they are involved in the I/O of the chip. The layout defines the positions of the cells (Figure 1).

The major objectives are chip area minimization and interconnection wire length minimization. Since the number of possible placements increases explosively with the number of blocks, even subsets of the problem have been shown to be NP-complete or NP-hard [7]. In this article, a hybrid genetic algorithm with two new heuristics is introduced for this problem. A genotype representation based on binary trees is used, and the genetic operators work directly on this tree structure.
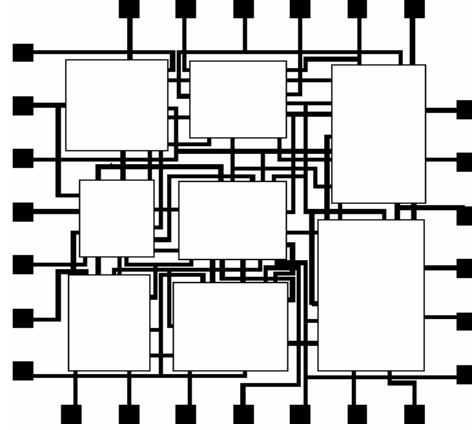
Fig. 1. The schematic representation of a VLSI macro-cell layout, which shows the position of eight cells, the routes for the signal nets, and the I/O pads.

2 Problem Description

Inputs of the placement problem are
• a set of blocks with fixed geometries and   fixed pin positions
• a set of nets specifying the interconnections between pins of blocks
• a set of pads (external pins) with fixed positions
• a set of user constraints, e.g., block positions/orientations, critical nets, if any

Given the inputs, the objective of the problem is to find the positions and orientations of each block, so that the chip area and interconnection wire length between blocks are minimized while satisfying all the given constraints. We take wire length into account simultaneously in the optimization process. Since it is impossible to calculate the exact wire length at this stage where detailed routing has not yet been carried out, we estimate the length of each net as one-half of the perimeter of the bounding box of the net.

The objective function, which measures the quality of the resulting placement, can be expressed as follows,

$$E = 1/(C_1 \text{ ChipArea} + C_2 \text{ WireLength})$$

where C_1, C_2 are the corresponding weights.

3 Genetic Layout Optimization

3.1 The Hybrid Genetic Algorithm

A Hybrid Genetic Algorithm is designed to use heuristics for improvement of offspring produced by crossover. Initial population is randomly generated. The offspring is obtained by crossover between two parents selected randomly. The layout improvement heuristics RemoveSharp and LocalOpt are used to bring the offspring to a local maximum. If fitness of the layout of the offspring thus obtained is greater than the fitness of the layout of any one of the parents then the parent with lower fitness is removed from the population and the offspring is added to the population. If the fitness of the layout of the offspring is lesser than that of both of its parent then it is discarded. For mutation a random number is generated within one and if it is less than the specified probability of the mutation operator a layout is randomly selected and removed from the population. Its layout is randomized and then added to the population. The algorithm works as below:

Step 1 :
 Initialize population randomly
Step 2 :
 Apply **RemoveSharp** algorithm to all layouts in the initial population
 Apply **LocalOpt** algorithm to all layouts in the initial population
Step 3 :
  Select two parents randomly
     Apply **Crossover** between parents and generate an offspring
     Apply **RemoveSharp** algorithm to offspring
     Apply **LocalOpt** algorithm to offspring
If Fitness(offspring) > Fitness (any one of the parents) then replace the weaker parent by the offspring
Step 4 :
 Mutate any one randomly selected layout from population
Step 5 :
 Repeat steps 3 and 4 until end of specified number of iterations.

3.2 Genotype Representation

 The phenotypic representation for the placement problems is basically the pattern that describes the position of the blocks. Binary slicing trees are well suited to represent placement patterns and have already been used in genetic algorithms [6]. During recombination, partial arrangements of blocks are transmitted from parents to offspring. The corresponding operation is the inheritance of subtrees from the parents. Encoding the tree in a string complicates this operation, since the string needs to be decoded into the slicing tree to execute the recombination, then re-

coded into an offspring chromosome afterwards. There is no reason for using a string encoding except for the analogy to the natural evolution process, where the genetic information is encoded in a DNA string.
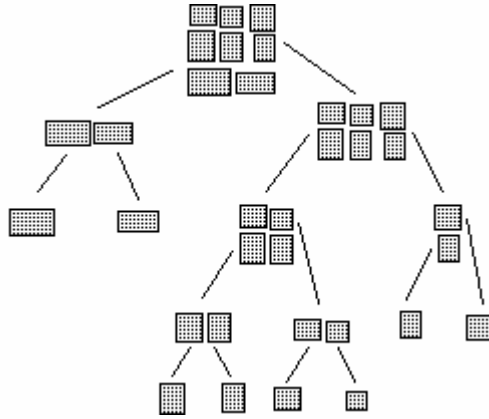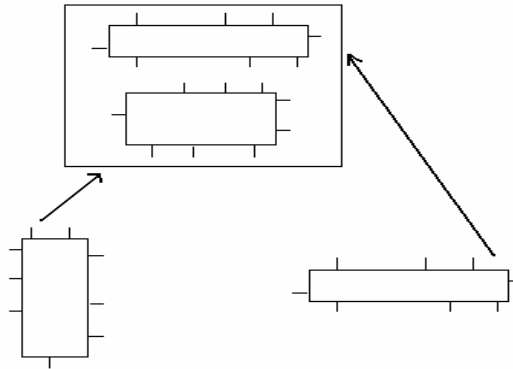

Fig. 2. The genotype

Fig. 3. The composition of a meta-block

When directly using the slicing tree as the genotype representation, further decoding or encoding the tree when applying genetic operators is avoided. The genotype is encoded as a binary slicing tree, which defines the relative placement of the cells (fig.2). It is composed in a bottom-up fashion. In each inner node two *blocks* (in the lowest level these are single cells) are joined to a *meta-block* (partial placement). In each meta-block the orientations of the combined blocks are fixed (fig.3). Therefore every tree describes several possible shapes for the corresponding layout, which enormously improves the performance of the GA. Blocks or

sub-patterns in a tree defining a layout, is always stacked vertically upon each other. The pattern characterized by the right successor of an inner tree node is always positioned on top of the pattern characterized by its left successor when combining both parts into a pattern or meta-block.

3.3 Genetic Operators

During the optimization process the placement of the blocks has to be changed. The genetic operators directly work on the tree-structure by combining subtrees of parents (crossover) and modifying the tree of an individual (mutation). The crossover operator takes two individuals (parent) out of which one offspring is composed by combining two subtrees, one from each parents. Unfortunately, these parts usually do not add up to a complete layout. After the combination of the two subtrees the redundant blocks are deleted and the missing blocks have to be added at random positions to the tree to ensure that the offspring finally represents a correct layout. Mutation operator modifies either by exchanging simple blocks or a block (leaf) with a meta-block (subtree) or by exchanging two meta-blocks. These cases represent the exchange of two cells, a cell with a partial layout, and the exchange of two partial layouts on the layout surface.

4 The RemoveSharp Algorithm

The RemoveSharp algorithm removes sharp increase in the wirelength due to a badly positioned macro-cell, which eventually decreases the fitness. The algorithm works as below:

Step 1 : A list (CONNECTIVITY-LIST) containing the m macro-cells highly connected to a selected macro-cell by signal nets is created.

Step 2 : RemoveSharp removes the selected macro-cell from the binary tree.

Step 3 : Now the selected macro-cell is reinserted in the binarytree either as left or right subtree to any one of the macro-cells in CONNECTIVITY-LIST and the fitness of the new layout is calculated for each case.

Step 4 : The genotype, which produces the highest fitness, is selected.

Step 5 : The above steps are repeated for each macro-cell in the binary tree.

4.1 Time Complexity of RemoveSharp

RemoveSharp removes a macro cell, inserts it at 2m (m is CONNECTIVITY-LIST size) different positions, and selects the best one. This is done for n macro cells. Using pointers the time taken for the functions, remove() for removing a block and insert() for inserting next to a specified macro cell is linear. The time complexity of the RemoveSharp algorithm is approximately linear as the basic operations are linear.

5   The LocalOpt Algorithm

The LocalOpt algorithm works as follows:
Step 1: Remove a subtree having q macro-cells from the binary tree.
Step 2: Compute the fitness of all possible arrangements of the n macro-cells (leaves) in the subtree.
Step 3: Find the one with maximum fitness value.
Step 4: Insert the optimized subtree at the position in the binary tree from where it is removed.

5.1   Time Complexity of LocalOpt

When LocalOpt is applied to a subtree only the links between the macro cells (attached to the leaf nodes) in the subtree and the leaf nodes in the subtree are changed. The binary tree structure of the subtree is not changed. The number of possible ways by which each one of the macro cells in the subtree can be assigned to a leaf node in the subtree is a constant (q!) for a given subtree size. For example it is 24 for subtree having 4 macro cells and 720 for subtree having 6 macro cells. Thus the time taken for applying LocalOpt to all the subtrees of same size is a constant. The Time Complexity of LocalOpt is hence k*O(N), where k is a constant. The constant k varies a little more than linearly with N due to the increase in time taken for finding the subtrees of specified size, when the number of macro cells in the problem (N) increases. Therefore the time taken for LocalOpt increases a little more than linearly for constant subtree size (q).
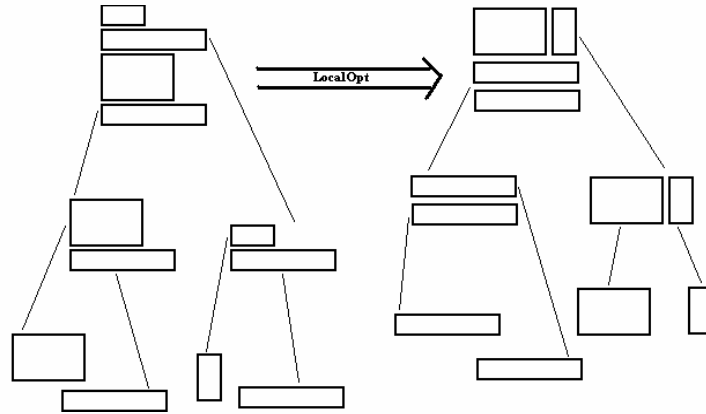

Fig. 4. LocalOpt applied to a subtree.

In fig.4 LocalOpt is applied to a subtree. It is obvious that the one obtained after applying LocalOpt (fig.4 right) results in a placement with higher fitness value.

6 Results

The hybrid genetic algorithm for the layout generation problem was tested on real-life circuits chosen from a benchmark suite that was released for design workshops in the early 90s and is often referenced in the literature as the *MCNC benchmark*s. They were originally maintained by *North Carolina's Microelectronics, Computing and Networking Cente*r, but are now located at the *CAD Benchmarking Laboratory (CBL)* at North Carolina State University. These benchmarks are standard problems in macro-cell layout, and the characteristics of the circuits are shown in Table 1. All the benchmark problems used in this paper can be downloaded from the URL: www.cse.ucsc.edu/research/surf/GSRC/MCNC/.

Table 1. The benchmark circuits for the macro-cell layout generation problem

Benchmark	Cells
Apte	9
Xerox	10
Hp	11

The algorithm is implemented in C++ and executed on a Pentium 300MHz workstation. The results are obtained within thirty seconds. The effects of various parameters and heuristics are analyzed; the best performance of HGA is found when the values of the parameters are set as below:

RemoveSharp (m)	:	5	LocalOpt(q)	:	6
Probability of			Population size	:	50
Mutation operator	:	0.02	Number of Iterations	:	10000

The classical layout problem is introduced in section 2; in which wirelength is also a factor in fitness value. But due to technological progress, technologies like Over-The-Cell routing are now used so there is no need to determine wirelength and to add routing space (through which wires are routed) to the layout [7]. Therefore in the fitness function introduced in section 2, C_1 is assigned one, and C_2 is assigned zero. It is difficult to fairly compare our algorithm with the other approaches reported, because they include routing space in their placements, which is not necessary any more. So the results of our algorithm are compared only with the results in [7].

In Table 2 our algorithm has been compared with Cluster Refinement for Block Placement. In case of Apte and Hp our algorithm has outperformed cluster refinement while in other case same result is obtained. Parallel Genetic Algorithm (PGA) can be designed using the new heuristics and better results can be obtained. It had been already shown that PGA generates better results than the sequential one [1,5].

Table 2.Comparison of HGA with Cluster Refinement Algorithm

Benchmark	HGA	Cluster Refinement
Apte	47.30	48.42
Xerox	20.30	20.30
Hp	9.39	9.575

7   Future Research

A parallelization of the genetic algorithm is planned and along with the strategy adaptation given in [5]. It might be ingenious to exchange or move large parts of the layout during the early stage of optimization, and doing only minor changes when the population converges to an optimum. This can become possible by adapting the frequencies of the mutation operator during the optimization. Further a gene-pool recombination operator [6,4] will be implemented which might replace the current crossover operator. For comparison a combinatorial optimization problem like the layout generation, the biologically motivated crossing of two parent chromosomes is likely to be less efficient. The construction of an offspring out of a pool of good building-blocks seems to be more suitable.

8   Conclusion

In this paper an approach has been presented to incorporate domain knowledge into a genetic algorithm, which is supposed to compute near-optimal solutions to VLSI Placement problem. The feasibility of the approach has been demonstrated by presenting performance results for benchmark instances. We find that the implementation of the two newly introduced heuristics result in near optimal solutions in all cases. These heuristics are simple, straightforward and easy to implement when compared to other algorithms. We believe this approach promises to be a useful tool in VLSI Design Automation.

References

1. Lienig J. (1997) A Parallel Genetic Algorithm for Performance Driven VLSI Routing. IEEE Transactions on Evolutionary Computation Vol. I. No.1 :29-39
2. Mazumder P., Rudnick E. (1999) Genetic Algorithm for VLSI Design, Layout and Automation. Addison-Wesley Longman Singapore Pte. Ltd., Singapore.
3. MCNC Benchmarks: www.cse.ucsc.edu/research/surf/GSRC/MCNC/

4. Schnecke V., Vornberger O (1996) A Genetic Algorithm for VLSI Physical Design Automation :In Proceedings of Second Int. Conf. on Adaptive Computing in Engineering Design and Control, ACEDC '96 26-28 Mar 1996, University of Plymouth, U.K., pp 53-58

5. Schnecke V., Vornberger O (1996) An Adaptive Parallel Genetic Algorithm for VLSI-Layout Optimization :In Proceedings of 4th Int. Conf. on Parallel Problem Solving from Nature (PPSN IV) 22-27 Sep 1996, Springer LNCS 1141, pp 859-868

6. Schnecke V., Vornberger O (1997) Hybrid Genetic Algorithms for Constrained Placement Problems. IEEE Transactions on Evolutionary Computation. Vol. I. No.4. :266-277

7. Xu, J., Guo P., et al. (1997) Cluster Refinement for Block Placement: In Proceedings of ACM-DAC California ,paper 47.4