# Effective Memetic Algorithms for VLSI Design = Genetic Algorithms + Local Search + Multi-Level Clustering

**Shawki Areibi** sareibi@uoguelph.ca
School of Engineering, University of Guelph, Guelph, Ontario, N1G 2W1, Canada

**Zhen Yang** zyang@uoguelph.ca
School of Engineering, University of Guelph, Guelph, Ontario, N1G 2W1, Canada

**Abstract**

Combining global and local search is a strategy used by many successful hybrid optimization approaches. Memetic Algorithms (MAs) are Evolutionary Algorithms (EAs) that apply some sort of local search to further improve the fitness of individuals in the population. Memetic Algorithms have been shown to be very effective in solving many hard combinatorial optimization problems. This paper provides a forum for identifying and exploring the key issues that affect the design and application of Memetic Algorithms. The approach combines a hierarchical design technique, Genetic Algorithms, constructive techniques and advanced local search to solve VLSI circuit layout in the form of circuit partitioning and placement. Results obtained indicate that Memetic Algorithms based on local search, clustering and good initial solutions improve solution quality on average by 35% for the VLSI circuit partitioning problem and 54% for the VLSI standard cell placement problem.

**Keywords**

Netlist Partitioning, Circuit Placement, Local Search, Genetic Algorithms, Memetic Algorithms, Multi-Level Clustering.

## 1 Introduction

Efficient optimization algorithms used to solve hard problems usually employ a hybrid of at least two techniques to find a near optimal solution to the problem being solved. The main motivation for hybridization in optimization practice is the achievement of increased efficiency (i.e. adequate solution quality in minimum time or maximum quality in specified time). From an optimization point of view, Memetic Algorithms combine global and local search by using Evolutionary Algorithms (EA) to perform exploration while the local search method performs exploitation. Memetic Algorithms have been shown to be very effective in solving many hard combinatorial optimization problems. This paper identifies the effect of local search, clustering and good initial solutions on the overall Memetic Algorithm by using the circuit partitioning and placement problems as a paradigm.

In the combinatorial sense, the circuit layout problem is a constrained optimization problem. We are given a circuit (usually a module-wire connection-list called a *netlist*) which is a description of switching elements and their connecting wires. Figure 1 shows a circuit consisting of several gates that need to be mapped onto the chip such that we minimize the interconnection between the modules. The inputs and outputs of the circuit in Figure 1-a are mapped onto I/O Pads in Figure 1-b. Each logic gate or module is called a cell[1] and the interconnection between the cells is established via the netlist as illustrated in the figure. We seek an assignment of geometric coordinates of

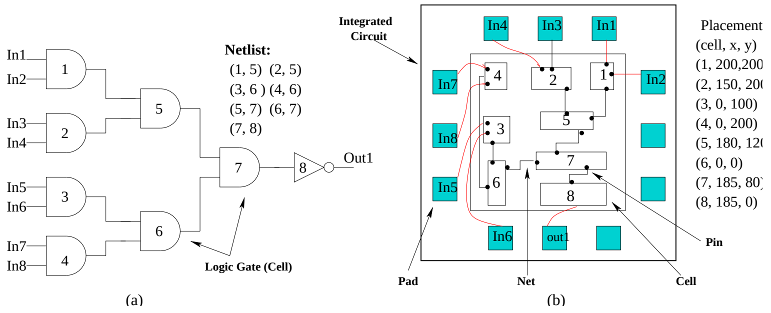


Figure 1: VLSI Physical Design Automation

the circuit components (in the plane or in one of a few planar layers) that satisfies the requirements of the fabrication technology (sufficient spacing between wires, restricted number of wiring layers, etc) and that minimizes certain cost criteria (i.e. power, area and delay).

## 1.1 Motivation

The interconnect effects have not been a serious concern in CMOS VLSI chips until recently, since the gate delays due to capacitive load components dominated the interconnect delay in most cases (KL03). However, with the introduction of deep sub-micron semiconductor technologies (transistor channel widths below 1$\mu$m), this picture has undergone rapid changes (RCN03). This fact is illustrated in Figure 2, where typical interconnect and gate delays are plotted for different technologies. It can be seen that for sub-micron technologies, both interconnect and gate delays decrease as the feature sizes decrease - but at different rates. This is because the gate delay usu-

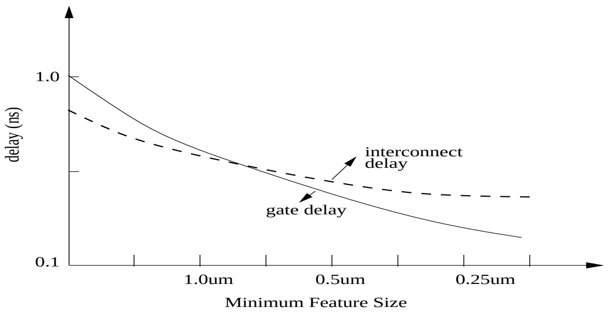


Figure 2: Interconnect and Gate Delay

ally decreases in sub-micron technologies while interconnect capacitance is independent of scaling[2]. The delay of a circuit, as well as the power dissipation and area, are

[1]We will use module or cell to refer to a logic gate in this paper.
[2]Reducing the transistor features.

dominated by interconnections between logical elements (i.e. transistors) in deep sub-micron regimes (RCN03). The most important influence of the increased interconnect delay is that the placement problem (which determines the location of devices) becomes very critical in today's VLSI design. Another important implication of decreasing devices and wire geometries is that the components in a circuit increase at a substantial rate. As a result, a partitioning/placement heuristic that produces excellent results for a small size problem may take weeks or months to obtain a good result. Obviously, a computationally expensive technique is often useless to the modern just-in-time fabrication mentality.

### 1.2 Contributions and Paper Organization

The main contributions of this paper are (i) the introduction of a new methodology for solving the VLSI circuit layout problem, (ii) integrating several techniques in the form of good initial solutions, local search and circuit clustering within a Genetic Algorithm paradigm, (iii) investigating balance between exploration and exploitation of the solution space (iv) improving the performance of Genetic Algorithms in general to solve the Circuit Partitioning/Placement problems by 35% and 54% respectively.

This paper is organized as follows: the iterative improvement and constructive techniques for circuit partitioning and placement are introduced in Section 2. The overall solution methodology is introduced in Section 3. The basic Genetic/Memetic algorithms along with some experimental results are then presented in Section 4. Section 5 introduces the clustering technique based Memetic algorithms and the corresponding experimental results. The paper concludes with some comments on how the local search and clustering technique impact on the performance of Genetic/Memetic Algorithms and possible future work.

## 2 Background

Practically all aspects of the layout problem as a whole are intractable (circuit partitioning, placement and global routing); that is, they are NP-hard (GJ79). Consequently, we have to resort to heuristic methods to solve very large problems. One of these methods is to break up the problem into subproblems (as seen in Figure 3, which are then solved one after the other. Almost always, these subproblems are NP-hard as well, but they are more amenable to heuristic solutions than is the entire layout problem itself. Each one of the layout subproblems is decomposed in an analogous fashion. In this way, we proceed to break up the optimization problems until we reach primitive subproblems. These subproblems are not decomposed further, but rather solved directly, either optimally (if an efficient polynomial-time optimization algorithm exists) or approximately if the subproblem is itself NP-hard or intractable, otherwise. The most common way of breaking up the layout problem into subproblems is first to do *logic partitioning* where a large circuit is divided into a collection of smaller modules according to some criteria. Following the circuit partitioning, the divided circuits are then placed on a layout surface. The main objectives of the circuit placement are to minimize the total chip area and the total estimated wire length for all the nets. Optimization of the chip area usage can fit more functionality into a given chip area. Optimization of the total estimated wire length can reduce the capacitive delays associated with longer nets and speed up the operation of the chip. Thus, the placement design process has a pronounced affect on the final chip performance. The final step in the VLSI circuit layout is to route the different nets connecting modules along the channels dedicated for this task (global/detailed routing phase).

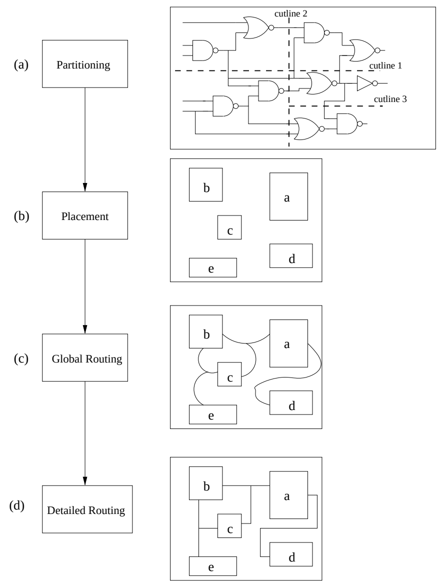


Figure 3: Physical Design Cycle (Circuit Layout)

## 2.1 Circuit Partitioning

Circuit partitioning is the task of dividing a circuit into smaller parts. It is an important aspect of layout for several reasons. Partitioning can be used directly to divide a circuit into portions that are implemented on separate physical components, such as printed circuit boards or chips. Here, the objective is to partition the circuit into parts such that the sizes of the components are within prescribed ranges and the complexity of connections (nets cut) between the components is minimized.

A standard mathematical model in VLSI layout associates a graph $G = (V, E)$ with the circuit netlist, where vertices in *V* represent modules, and edges in *E* represent signal nets. The netlist is more generally represented by a *hypergraph* $H = (V, E')$, where hyperedges in $E'$ are the subsets of V contained by each net (since nets often are connected to more than two modules). In this formulation, we attempt to partition a circuit with $n_m$ modules and $n_n$ nets into $n_b$ blocks containing approximately $\frac{n_m}{n_b}$ modules each; (i.e. we attempt to equi-partition the V modules among the $n_b$ blocks), such that the number of uncut nets in the $n_b$ blocks is maximized.
We define:

$$x_{ik} = \begin{cases} 1 & \text{if module } i \text{ is placed in block } k \\ 0 & \text{otherwise} \end{cases}$$

$$y_{jk} = \begin{cases} 1 & \text{if net } j \text{ is placed in block } k \\ 0 & \text{otherwise} \end{cases}$$

The linear integer programming (LIP) model of the netlist partitioning problem is given by maximizing the number of uncut nets in each block;

$$Max \sum_{j=1}^{n_n} \sum_{k=1}^{n_b} y_{jk} \tag{1}$$

s.t. (i) Module placement constraints:

$$\sum_{k=1}^{n_b} x_{ik} = 1, \quad \forall_i = 1, 2, \ldots, n_m$$

(ii) Block size constraints:

$$\sum_{i=1}^{n_m} x_{ik} \leq \frac{n_m}{n_b}, \quad \forall_k = 1, 2, \ldots, n_b$$

(iii) Netlist constraints:

$$y_{jk} \leq x_{ik}, \ where \quad \begin{array}{l} 1 \leq j \leq n_n \\ 1 \leq k \leq n_b \\ i \in Net\ j \end{array}$$

(iv) 0-1 constraints:

$$\begin{array}{lll} x_{ik} \in \{0,1\}, & 1 \leq i \leq n_m; & 1 \leq k \leq n_b \\ y_{jk} \in \{0,1\}, & 1 \leq j \leq n_n; & 1 \leq k \leq n_b \end{array}$$

The *module placement constraints* ensure that each module can be placed in only one block. The *block size constraints* guarantee that each block will not exceed a certain limit of modules. Finally, the *netlist constraints* determine if a net (wire) *j* is placed entirely in block *k* or if it is not. In problem (LIP) we maximize the number of uncut nets in the $n_b$ blocks. This is equivalent to the netlist partitioning problem where we minimize the number of wires connecting the $n_b$ blocks.

The heuristic algorithms for partitioning problems can be classified as being *constructive* or *iterative*. Constructive algorithms determine a partitioning from the graph describing the circuit or system, whereas iterative methods aim at improving the quality of an existing partitioning solution.

#### 2.1.1 Iterative Techniques for Circuit Partitioning

To date, iterative improvement techniques that make local changes to an initial partition are still the most successful algorithms in practice. The advantage of these heuristics is that they are quite robust. In fact, they can deal with netlists as well as arbitrary vertex weights, edge costs, and balance criteria.

Kernighan and Lin (KL70) described a successful heuristic procedure for graph partitioning which became the basis for most module interchange-based improvement partitioning heuristics used in general. Their approach starts with an initial bisection and then involves the exchange of pairs of vertices across the cut of the bisection to improve the cut-size. The algorithm determines the vertex pair whose exchange results in the largest decrease of the cut-size *or* in the smallest increase, if no decrease is possible. A pass in the Kernighan and Lin algorithm attempts to exchange all vertices on both sides of the bisection. At the end of a pass the vertices that yield the best cut-size are the only vertices to be exchanged.

Fiduccia and Mattheyses (FM82) (usually referred to as the FM technique) modified the Kernighan and Lin algorithm by suggesting to move one cell at a time instead of exchanging pairs of vertices, and they also introduced the concept of preserving balance in the size of blocks. The FM method reduces the time per pass to linear in the size of the netlist (i.e. O(p), where p is the total number of pins) by adapting a

```
current_solution ← initial_solution
current_cost ← evaluate(current_solution)
Repeat
        initialize partition
        While (can_move(modules))
            choose cell with highest gain
            update gains of all cells
            if ($current\_gain > previous\_gain$)
                bestgain = current_gain
        end while
        move nodes pointed to by bestgain_ptr
        if (no improvement)
            ++noimp_counter
Until((pass > MaxPass) OR
            (noimp > MaxNoImp))
```

Figure 4: Fiduccia Mattheyses Algorithm

single-cell move structure, and a gain bucket data structure that allows constant-time selection of the highest-gain cell and fast gain updates after each move. Figure 4 shows the basic FM algorithm used for circuit partitioning (FM82). Sanchis (San89) uses the above technique for multiple way network partitioning. Under such a scheme, we should consider all possible moves of each free cell from its home block to any of the other blocks, at each iteration during a pass the best move should be chosen. As usual, passes should be performed until no improvement in cutset size is obtained.

A modified implementation of the Sanchis iterative improvement heuristic called Simple Dynamic Hill Climbing (SDHC) (Are00) can also be used. It is characterized by the ability for local optimum to escape, which usually causes simple descent algorithms to terminate, by dynamically taking a different direction of a steepest ascent. In SDHC, after the termination of the Sanchis heuristic, the technique considers all possible moves of each free cell from its home block to any of the other blocks, such that the value of the cut-size is increased. This is done such that the direction of the upward slope is different than the last pass performed. The main objective of SDHC is to explore small regions effectively in relatively short periods of time.

### 2.1.2 Constructive Techniques

In general, node interchange methods are greedy or local in nature and get easily trapped in local minima. More importantly, it has been shown that interchange methods fail to converge to "optimal" or "near optimal" partitions unless they initially begin from "good" partitions. Sechen (SC88) shows that over 100 trials or different runs (each run beginning with a randomly generated initial partition) are required to guarantee that the best solution would be within twenty percent of the optimum solution. In order for interchange methods to converge to "near optimal" solutions they have to initially begin from "good" starting points (Are00). Constructive partitioning approaches are mainly based on clustering (AV96; DD96), placement-based partitioning (RDJ94), mathematical programming or network flow computations.

GRASP is a greedy randomized adaptive search procedure that has been successful in solving many combinatorial optimization problems efficiently (FRS94). Figure 5 shows a pseudo-code of the GRASP heuristic. The algorithm starts with a construction phase followed by a local improvement phase. The GRASP implementation terminates after a certain number of phases or runs have passed. The construction phase as shown in Figure 5(B) is iterative, greedy and adaptive. It is *iterative* because the initial solution is built by considering one element at a time. The choice of the next element to be added is determined by ordering all elements in a list. The list of the best candidates is called the restricted candidate list (RCL). It is *greedy* because the addition of each element is guided by a greedy function. The construction phase is *randomized* by allowing the selection of the next element added to the solution to be any element in the RCL. Finally, it is *adaptive* because the element chosen at any iteration in a construction is a function of those previously chosen. The improvement phase typically consists of a local search procedure as shown in Figure 5(C). A more sophisticated local search based on Tabu Search can be implemented instead of the simple local search procedure.

```
(A) MAIN_GRASP()
  1. Read_Circuit_NetList();
  2. do
     3. Construction_Phase(Greedy,Rand,Adapt)
     4. Local_Improvement_Phase(Initial Solution)
     5. Store_Best_Solution(Previous Solutions)
     6. While not Done
  7. Report_Best_Solution()
(B) Construction_Phase()
  1. While (construction not done)
     2. Greedy: Create Candidate List (RCL)
     3. Random: Module = Select from RCL
     4. Adaptive: Add new element to solution
     5. Feasibility: Check Feasibility of Solution
  6. EndWhile
(C) Local_Improvement_Phase()
  1. Read Initial Solution
  2. While (local optimum not reached)
     3. NewSolution = Local_Changes(Solution)
  4. EndWhile
  5. Return_Best_Solution()
```

Figure 5: GRASP (Greedy Adaptive Search)

### 2.2 Circuit Placement

Usually, a circuit is represented by a hypergraph $G(V, E)$, where the vertex set $V = \{v_1, v_2, \cdots, v_n\}$ represents the nodes of the hypergraph (set of cells to be placed), and $E = \{e_1, e_2, \cdots, e_m\}$ represents the set of edges of the hypergraph (set of nets connecting the cells). The two dimensional placement region is represented as an array of legal placement locations. The hypergraph is transformed into a graph (a hypergraph with all hyperedge sizes equal to 2) via a clique model for each net. Each edge $e_j$ is an ordered pair of vertices with a non-negative weight $w_j$ assigned to it. The placement task seeks to assign all cells of the circuit to legal locations such that cells do not overlap (as explained earlier in Figure 1. Each cell $i$ is assigned a location $(x_i, y_i)$ on the XY-plane. The cost of an edge connecting two cells $i$ and $j$ with locations $(x_i, y_i)$ and $(x_j, y_j)$ is computed as the product of the squared $l_2$ norm of the difference vector $(x_i - x_j, y_i - y_j)$ and the weight of the connecting edge $w_{ij}$. The total cost, denoted $\phi(x, y)$, can then be given as the sum of the cost over all edges; i.e.:

$$\phi(x,y) = \sum_{1 \le i < j \le N} w_{ij}[(x_i - x_j)^2 + (y_i - y_j)^2] \tag{2}$$

Minimizing (2) produces a placement with a great amount of overlap among the cells because it attracts cells sharing common nets together[3]. Formulation (2) can be written in matrix form as:

$$\phi(x,y) = \frac{1}{2}\mathbf{x}^T\mathbf{C}\mathbf{x} + \mathbf{d}_x^T\mathbf{x} + \frac{1}{2}\mathbf{y}^T\mathbf{C}\mathbf{y} + \mathbf{d}_y^T\mathbf{y} + \mathbf{t} \tag{3}$$

where vectors $\mathbf{x}$ and $\mathbf{y}$ denote the coordinates of the $N$ movable cells; matrix $\mathbf{C}$ is the Hessian matrix; vectors $\mathbf{d}_x^T$ and $\mathbf{d}_y^T$ and the constant term $\mathbf{t}$ result from the contributions of the fixed cells. Normally the first moment constraints are added to force the distribution of the cells to be uniform around the center of the placement area. It follows that the quadratic placement model is given as:

$$\begin{aligned} &\text{Min } \phi(x,y) \\ &\text{s.t. } A_x x = b_x \\ &\quad A_y y = b_y \\ &\quad l_x \le x_i \le u_x \\ &\quad l_y \le y_i \le u_y \end{aligned}$$

where $A_x$ and $A_y$ are $q \times n$ matrices; $q$ is the number of regions into which the placement area has been partitioned. The $q \times 1$ vectors $b_x$ and $b_y$ represent the centers of the $q$ regions. The parameters $l_x$, $u_x$, $l_y$ and $u_y$ are lower and upper bounds on the $x$ and $y$ coordinates of the cells. Clearly, the above optimization problem can be split into two 1-dimensional subproblems and each subproblem can then be solved independently (YA02).

Depending on the input, the placement algorithms can be classified into two major groups, referred to as the *constructive placement* and *iterative improvement* methods. Usually, constructive placement algorithms include *random placement, cluster growth, partitioning-based placement* (GJ79), *numerical optimization*, and *branch and bound techniques* (RDJ94). There are two classes of iterative improvement placement methods: *deterministic* and *stochastic heuristics*. A deterministic heuristic interchanges randomly selected pairs of modules and only accepts the interchange if it results in a reduction in cost (YA02). On the contrary, a stochastic heuristic not only accepts the possible perturbation that results in cost reduction but also uses some "randomness" to accept some poor solutions, which allows the heuristic to avoid the local-optimal and explore the solution space more effectively.

#### 2.2.1 Tile Based Iterative Improvement

A good placement heuristic should consider both quality and computational efficiency issues. The quality of the placement is essential for the performance of the final circuit whereas computational efficiency is essential to shorten the design procedure. The Tile based iterative heuristic is a kind of algorithm that produces a good solution in a reasonable amount of time (ATV01). In the Tile based approach overlapping tiles or windows (ATV01) are introduced to localize the arrangement of cells throughout the placement area. The introduction of tiles is useful for restricting the rearrangement of cells. Each tile contains a small subset of the cells. Furthermore, cells may belong

[3]We refer to formulation (2) as the QP (Quadratic Programming) formulation.

to more than one tile due to the overlap which exists between tiles. The Tile iterative approach works as follows. A tile is selected, and a list of cells within the tile is generated. Subsequently, cells within the selected tile are rearranged in some fashion. In rearranging the cells, the cells are restricted to positions within the tile boundaries, which keeps cells close to their original positions. The computational effort required to find improved cell positions is reduced, since the search for improved cell positions is restricted to positions within the tile boundaries. Since tiles overlap, cells near the boundaries of a tile may be permitted to move between tiles. During one pass, tiles are selected randomly, and only once during each pass. After each pass, the quality of the placement is evaluated and compared to placements generated by previous passes. The algorithm terminates when either a maximum number of passes is exceeded, or when the improvement in the placements over a number of consecutive passes is negligible. Figure 6 summarizes the Tile approach.

```
1. Input: initial placement $s$;
        estimate_wirelength(initial_placement)
2. Repeat
        select cells i and j;
        determine_cell_shifts();
        estimate_wirelength(new_placement)
        If $\triangle$Total_wirelength $\leq 0$ do
         update_ placement();
3. Until attempted_moves $\geq$ max_attempts;
4. Output Best placement found so far
```

Figure 6: A Tile-based Algorithm

### 2.2.2 Cluster-Seed Based Constructive Technique

Cluster-Seed placement is one of the cluster growth placement algorithms (YA02). Cluster growth placement is a bottom-up method that operates by selecting modules and adding them to a partial placement (KP86). Normally, it contains two functions: (i) *selection function* and (ii) *placement function*. The selection function determines the order in which the unplaced modules are included in the placement. The placement function decides the best position for those modules (KP86).

The Cluster-Seed based placement algorithm is described in Figure 7. First, a seed placement is determined by selecting the pads in natural order and then placing all unplaced modules that are directly connected with these pads. Next, other unplaced modules are selected one at a time in order of their connectivity to the placed modules (highly connected first) and fixed at a vacant position close to the placed modules so that the total wire length is minimized.

## 2.3 Circuit Clustering

As the complexity of VLSI circuits increases, a hierarchical design approach becomes essential to shorten the design period (HK92). Circuit clustering plays a fundamental

```
1. Set Total_mods, placed_mods;
2. Place the modules connecting with the
   pads directly and update placed_mods;
3. While (placed_mods $\neq$ Total_mods)
      If (all the pads on the top side)
          Top_pads_placement();
      Elseif (all the pads on the bottom side)
          Bottom_pads_placement();
      Else
          Normal_pads_placement();
      update placed_mods
   End While
5. Return the legalized placement solution.
```

Figure 7: A Cluster-Seed Algorithm

role in hierarchical designs. Identifying highly connected components in the netlist can significantly reduce the complexity of the circuit and improve the performance of the design process. Figure 8 illustrates the procedure utilized to combine highly connected modules to form super clusters for both circuit partitioning and placement. Following clustering, several efficient heuristic techniques can be used to minimize the *nets cut* and/or place modules efficiently on the chip. The circuit has to be transformed again to its original representation and therefore it is *flattened* (i.e de-clustered). Section 5 will further demonstrate the effectiveness of this technique in solving both problems using a Memetic Algorithm.

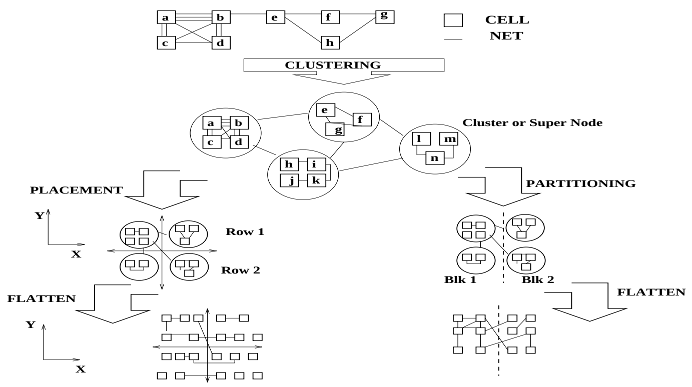


Figure 8: Overview of clustering

| Circuit | Cells | Nets | Pins | Pads | Rows | Cell Degree | | | Net Size | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | MAX | $\overline{x}$ | $\sigma$ | MAX | $\overline{x}$ | $\sigma$ |
| Fract | 125 | 147 | 462 | 24 | 6 | 7 | 3.1 | 1.6 | 17 | 3.1 | 2.2 |
| Prim1 | 833 | 902 | 2908 | 81 | 16 | 9 | 3.4 | 1.2 | 18 | 3.2 | 2.5 |
| Struct | 1888 | 1920 | 5471 | 64 | 21 | 4 | 2.8 | 0.6 | 17 | 2.8 | 1.9 |
| Ind1 | 2271 | 2192 | 7743 | 814 | 15 | 10 | 3.4 | 1.1 | 318 | 3.5 | 9.0 |
| Prim2 | 2907 | 3029 | 18407 | 107 | 28 | 9 | 3.7 | 1.5 | 37 | 3.7 | 3.8 |
| Bio | 6417 | 5742 | 21040 | 97 | 46 | 6 | 3.2 | 1.0 | 861 | 3.6 | 20.8 |
| Ind2 | 12142 | 13419 | 48158 | 495 | 72 | 12 | 3.8 | 1.8 | 585 | 3.5 | 10.9 |
| Ind3 | 15057 | 21808 | 65416 | 374 | 54 | 12 | 4.3 | 1.4 | 325 | 2.9 | 3.2 |
| Avq.s | 21854 | 22124 | 76231 | 64 | 80 | 7 | 3.4 | 1.4 | 4042 | 3.4 | 53.3 |
| Avq.l | 25144 | 25384 | 82751 | 64 | 86 | 7 | 3.2 | 1.2 | 4042 | 3.2 | 49.8 |

Table 1: Benchmarks Used as Test Cases

### 2.4 VLSI Benchmarks

For the partitioning and placement problems, the quality of solutions obtained are based on a set of hypergraphs that are part of the widely used ACM/SIGDA (RP87) circuit partitioning/placement benchmarks suite. The characteristics of these hypergraphs are shown in Table 1. The second column of the table shows the number of cells within the circuit. The third column presents the number of nets connecting the cells within the benchmarks. The total number of pins (i.e connections) within the circuit is summarized in column four. The fifth column indicates the pads (i.e I/O connections) that connect the circuit to the outside world. The sixth column gives the number of rows where the cells are to be placed (exclusively for the circuit placement problem). The last two columns summarize the statistics of the circuit (i.e connectivity). The proposed optimization techniques are implemented in the 'C' programming Language on a Sun Ultra10 workstation.

## 3 Solution Methodology

The traditional approach in partitioning and placement is to construct an *initial solution* by using constructive heuristic algorithms. A *final solution* is then produced by using iterative improvement techniques where a modification is usually accepted if a reduction in cost occurs, otherwise it is rejected. Constructive heuristic algorithms pro-

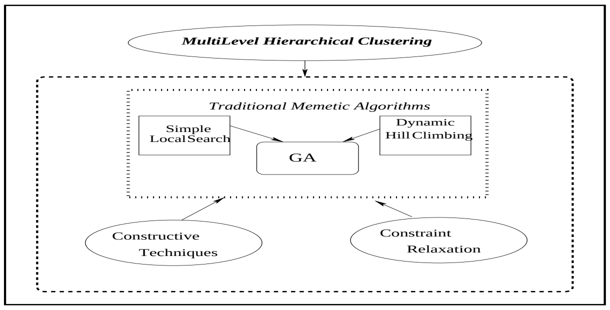


Figure 9: Memetic Algorithm Approach for VLSI Design

duce an initial solution from scratch. It takes a negligible 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 improve 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. Figure 9 summarizes the overall approach used in this paper to solve hard VLSI combinatorial optimization problems. 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 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 (e.g size constraint for circuit partitioning) with intermediate relaxation mechanism to further enhance the solution quality.

### 3.1 Exploration versus Exploitation

Exploration is the process of visiting entirely new regions of a search space (new regions where the gain can be high) to determine if anything promising may be found. Exploitation on the other hand concentrates on previously visited points to maximize the gain (i.e the determination of which places might be profitable to visit next). A purely random search is good at exploration whereas a purely hill-climbing method is good at exploitation. Combinations of these two strategies can be quite effective, but it is difficult to know where the best balance is set. One of the main objectives in implementing any Memetic Algorithm is the means of achieving both techniques during the search. In this paper we attempt to illustrate the best balance to achieve good results for both circuit partitioning and placement. It is important to understand that injecting constructive initial solutions within a population is a form of local search. Also, the concept of clustering that will be utilized in this paper attempts to smooth the landscape being searched and therefore can be considered a different form of iterative improvement embedded within the Memetic Algorithm. Results introduced in Section 4 and 5 will illustrate how this concept of utilizing exploration and exploitation can be used to produce efficient results for both problems.

## 4 Evolutionary Algorithms

Evolutionary Algorithms **(EA's)** are a class of optimization algorithms that seek improved performance by sampling areas of the parameter space that have a high probability for leading to good solutions (Mic92). As an optimization technique, Genetic Algorithms simultaneously examine and manipulate a set of possible solutions. The power of GA's comes from the fact that the technique is robust, and can deal successfully with a wide range of problem areas, including those which are difficult for other methods to solve. GA's are not guaranteed to find the global optimum solution to a problem, but they are generally good at finding "acceptably good" solutions to problems. Another drawback of Genetic Algorithms is that they are not well suited to perform finely tuned search, but on the other hand they are good at exploring the solution space since they search from a set of designs and not from a single design.

### 4.1 Genetic-based Partitioning Algorithm

One way to represent the partitioning problem (as seen in Figure 10) is to use *group-number encoding* where the $j^{th}$ integer $i_j \in \{1, \ldots, k\}$ indicates the group number assigned to object *j*. This representation scheme creates the possibility of applying stan-

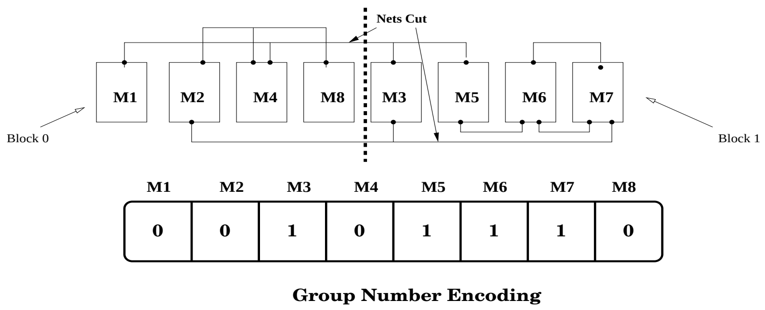


Figure 10: Chromosome Representation for Circuit Partitioning

dard operators (Mic92). However an offspring may contain less than *k* groups; moreover, an offspring of two parents, both representing feasible solutions may be infeasible, since the constraint of having equal number of modules in each partition is not met. In this case either special *repair heuristics* are used to modify chromosomes to become feasible, or *penalty functions* that penalize infeasible solutions, are used to eliminate the problem. The penalty function approach may lead to good solutions but is computationally expensive. Instead we use the repair heuristics as a means to obtain feasible solutions.

Figure 11 illustrates a Genetic Algorithm implementation for circuit partitioning. The GA starts with 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 the partitioning-specific fitness function. The 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 as seen in Figure 12b. Thus each offspring inherits a combination of features from both parents. The next step is *mutation* (as illustrated in Figure 12a) where an incremental change is made to each member of the population, with a small probability. This ensures that the 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. Our empirical study, strongly supports using multi-point crossover over the one-point crossover technique. A 3-point and 4-point crossover works best for our circuit partitioning problem. In this implementation we have used the Roulette Wheel parent selection method which is conceptually one of the simplest stochastic selection technique. Our generation replacement technique is based on replacing the most inferior member in a population by new offsprings.

## 4.2 Memetic Based Partitioning Algorithms

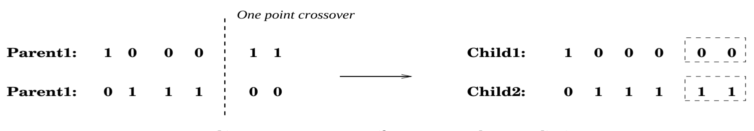
Genetic Algorithms are not well suited for fine-tuning structures which are close to optimal solutions (Gol89). Incorporation of local improvement operators into the recombination step of a Genetic Algorithm is essential if a competitive Genetic Algorithm is desired. Memetic Algorithms (MAs) apply a separate local search process to refine

```
1. Encode Solution Space for circuit
2.(a) set pop_size, max_gen, gen=0;
  (b) set cross_rate, mutate_rate;
3. Initialize Population.
4. While  max_gen ≥ gen
        Evaluate Fitness (# of cuts)
        For (i=1 to pop_size)
          Select (mate1,mate2)
          if (rnd(0,1) ≤ cross_rate)
            child = Crossover(mate1,mate2);
          if (rnd(0,1) ≤ mutate_rate)
            child = Mutation();
          Repair child if necessary
        End For
        Add offsprings to New Generation.
        gen = gen + 1
   End While
5. Return best chromosomes.
```

Figure 11: A Genetic Algorithm for VLSI Circuit Partitioning

| Old Chromosome | | | | Random Numbers | | | | New Bit | New Chromosome | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | .801 | .102 | .266 | .373 | – | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | .120 | .096 | .005 | .840 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | .760 | .473 | .894 | .001 | 1 | 0 | 0 | 1 | 1 |

(a) Mutation Operator



(b) Crossover Operator (for group number encoding)

Figure 12: Crossover and Mutation Operators for Circuit Partitioning

individuals (i.e improve their fitness by hill-climbing). Memetic Algorithms have been shown to be very effective for many combinatorial optimization problems, including the quadratic assignment problem (QAP), traveling salesman problem (TSP) and many others. In particular, the relative advantage of MAs over EAs is quite consistent on complex search spaces. In the next few subsections we present several approaches of integrating Genetic Algorithms with local search heuristics to effectively design robust Memetic Algorithms for circuit partitioning and placement problems.

### 4.2.1 A Simple Memetic Implementation

Figure 13 shows a simple implementation of a Memetic algorithm based on the Genetic Algorithm introduced in Section 4.1. We use a simple variation of the Fiduccia and Mattheyses (FM) heuristic (San89). The original FM heuristic has several passes after which the heuristic terminates as presented in Section 2. In the local optimization phase, a few passes are allowed[4], furthermore a restriction on the number of modules to be moved is set to a certain value. It is to be noted that if local optimization is not strong enough to overcome the inherent disruption of the crossover, stronger local optimization is needed.

```
1. Encode Solution Space for circuit paritioning
2.(a) set pop_size, max_gen, gen=0;
   (b) set cross_rate, mutate_rate;
3. Initialize Population.
4. While(Gen < Gensize)
      Apply GA
         Apply FM Local Search to Population
EndWhile /* end of a run */
Apply Final Local Search to Best Chromosome
```

Figure 13: A Partitioning based Memetic Algorithm

### 4.2.2 A Parameter Relaxation Approach

A standard technique in solving optimization problems involves relaxation of one or more problem parameters. In this section we illustrate the use of this technique in Memetic Algorithms. The partitioning problem for any given objective is inherently a constraint-driven one. The most common constraint is the balance ratio of total sizes that the two subsets of a partition must satisfy (DT97). Other frequently encountered partitioning constraints include limits on the sizes or timing minimization. This balance-constrained partitioning problem can be stated as follows. Given a netlist **G** which describes cell connectivities and cell sizes in a circuit, construct k sub-circuits of **G** with a balance ratio of $r_1 : r_2 : ... : r_k$ and an acceptable tolerance of $\pm t$ such that some objective functions are optimized. In an attempt to satisfy the balance-ratio, however, movement of larger cells and clusters can be restricted thus leading to sub-optimal results. One of the fundamental constraints in bi-partitioning is the required balance in the sizes of $V_1$ and $V_2$. Assuming $|V_1| \leq |V_2|$, the constraint can be stated as $r - t \leq \frac{|V_1|}{|V_1 + V_2} \leq r + t$. During the search phase, a temporary violation of the size constraint is allowed, however at the end, size constraints are satisfied and fully restored by a greedy algorithm. Relaxed partitioning helps in removing natural clusters that straddle the cut-line by blocks of unidirectional moves that temporary violate the

[4]The number of passes depends on the fitness of each chromosome in the population

balance constraint; in a non-relaxed partitioner, these clusters can get locked in the cut-set due to alternating moves in the two directions. The proposed Memetic algorithm as seen in Figure 14 starts with the GA technique followed by SDHC with and without relaxation of size constraints.

```
While(Gen < Gensize)
   Apply GA
   If Relax Size Constraint
      Relax Size of Required Partitions
      If Apply Local Search
         Apply FM Local Search to Population
      Else
         Apply SDHC to Population
      Enforce Size Constraint Greedly
   Else
      Apply SDHC or FM to Population
EndWhile /* end of a run */
```

Figure 14: A Memetic Algorithm with Relaxation

The parameters used for the GA are similar to those explained in Section 4.1. If the size constraints are relaxed, then the algorithm applies either FM (Sanchis multi-way partitioning) or SDHC (simple dynamic hill climbing) on the population for a limited number of passes (determined by the user). Size constraints are enforced at the end of the algorithm using a greedy heuristic.

### 4.2.3 Computational Results for Memetic Algorithm Implementation

Tables 2 and 3 compare the performance of a pure Genetic Algorithm to a memetic technique that combines GA with simple local search techniques and also a more advanced hill climbing technique. All results are the average of 10 runs[5].

The first column of each table presents the results based on a pure Genetic Algorithm with a population size of **50**, crossover rate of **99%**, mutation rate of **0.36%** and generation size of **10**. The second column in each table is based on a simple integration of Genetic Algorithms with a local search heuristic (GA-II). In each generation a few passes of local search are applied to each chromosome. The third column presents a Memetic Algorithm (GA-FI) where a local search approach is applied to all chromosomes in only the last two generations. The fourth column presents an implementation that combines (GA-II) with (GA-FI) (i.e strong local search). The last column of each table gives results based on an integrated Genetic Algorithm with a dynamic hill climbing technique with size constraints relaxed as explained above. Results obtained for four-way partitioning indicate that for most circuits the amount of improvement achieved is on average 16% for the GA-DHC implementation at the expense of increased CPU time. It is interesting to notice that the GA-II implementation where local search is applied to all the chromosomes in every generation does not achieve as good results as expected. This is due to premature convergence of the solutions because of the strong local search applied to the population. Table 4 is identical to Table 3 except that 25% of the population is injected with good initial solutions based on GRASP. The quality of solutions obtained by GA-GR (Genetic Algorithm with GRASP) alone enhances the

[5]Fractional parts excluded due to large number of nets within a circuit.

| Memetic Algorithms With Random Initial Solutions | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Circuit** | **Pure GA** | | **GA-II** | | **GA-FI** | | **GA-IFI** | | **GA-DHC** | |
| | Cuts | Time | Cuts | Time | Cuts | Time | Cuts | Time | Cuts | Time |
| Fract | 46 | 0:01 | 49 | 0:01 | 11 | 0:01 | 23 | 0:01 | 11 | 0:01 |
| Prim1 | 87 | 0:06 | 87 | 0:07 | 71 | 0:07 | 87 | 0:06 | 68 | 0:07 |
| Struct | 77 | 0:13 | 65 | 0:15 | 55 | 0:16 | 51 | 0:15 | 45 | 0:17 |
| Prim2 | 196 | 0:24 | 198 | 0:27 | 171 | 0:29 | 190 | 0:27 | 171 | 0:32 |
| Ind1 | 78 | 0:17 | 76 | 0:19 | 65 | 0:20 | 75 | 0:19 | 57 | 0:22 |
| Bio | 211 | 0:46 | 206 | 0:51 | 195 | 0:54 | 204 | 0:51 | 115 | 1:00 |
| Ind2 | 299 | 2:09 | 286 | 2:21 | 278 | 2:31 | 261 | 2:20 | 278 | 2:47 |
| Ind3 | 555 | 3:15 | 491 | 3:34 | 448 | 3:52 | 491 | 3:34 | 415 | 4:15 |
| Avq.s | 511 | 3:35 | 516 | 3:55 | 483 | 4:06 | 512 | 3:57 | 483 | 4:38 |
| Avq.l | 486 | 4:08 | 526 | 4:30 | 467 | 4:50 | 488 | 4:34 | 467 | 5:21 |
| **Total** | 2546 | 894 | 2500 | 980 | 2244 | 1046 | 2382 | 984 | 2110 | 1160 |
| %Imp | 0% | 0% | +2% | -8% | +12% | -14% | +6% | -9% | +17% | -22% |

Table 2: Performance of 2-Way GA/MA Partitioning (Random Initial)

| Memetic Algorithms With Random Initial Solutions | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Circuit** | **Pure GA** | | **GA-II** | | **GA-FI** | | **GA-IFI** | | **GA-DHC** | |
| | Cuts | Time | Cuts | Time | Cuts | Time | Cuts | Time | Cuts | Time |
| Fract | 67 | 0:01 | 70 | 0:01 | 42 | 0:02 | 48 | 0:01 | 31 | 0:02 |
| Prim1 | 162 | 0:07 | 154 | 0:08 | 129 | 0:09 | 145 | 0:08 | 129 | 0:11 |
| Struct | 253 | 0:18 | 240 | 0:21 | 192 | 0:21 | 219 | 0:19 | 181 | 0:23 |
| Prim2 | 447 | 0:29 | 400 | 0:32 | 370 | 0:37 | 348 | 0:32 | 370 | 0:38 |
| Ind1 | 140 | 0:21 | 134 | 0:24 | 101 | 0:27 | 104 | 0:23 | 100 | 0:28 |
| Bio | 285 | 1:03 | 307 | 1:08 | 266 | 1:17 | 284 | 1:09 | 266 | 1:23 |
| Ind2 | 1287 | 2:55 | 1340 | 3:11 | 894 | 3:48 | 753 | 3:15 | 894 | 4:02 |
| Ind3 | 1581 | 4:18 | 1518 | 4:45 | 1360 | 5:33 | 1441 | 4:48 | 1360 | 5:46 |
| Avq.s | 1079 | 4:57 | 1070 | 5:23 | 995 | 6:16 | 989 | 5:26 | 995 | 6:37 |
| Avq.l | 1071 | 5:38 | 1070 | 6:11 | 1019 | 7:17 | 1002 | 6:14 | 1019 | 7:39 |
| Total | 6372 | 1207 | 6303 | 1324 | 5368 | 1547 | 5333 | 1335 | 5345 | 1629 |
| %Imp | 0% | 0% | +1% | -8% | +16% | -21% | +16% | -10% | +16% | -26% |

Table 3: 4-Way GA/MA Random Partitioning (Random Initial)

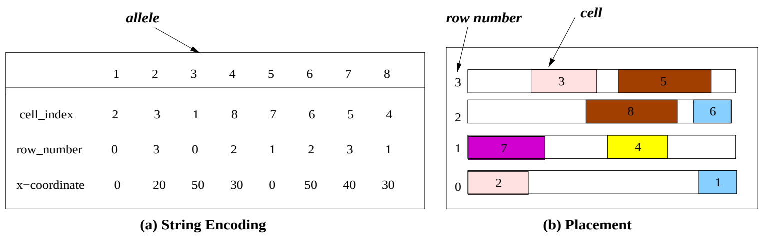
quality of the solution by 12% over a GA implementation that utilizes an initial random population. Solutions obtained by GA-GR-DHC which utilizes a dynamic hill climbing technique with relaxed size constraints improve on average by 20% over a simple Genetic Algorithm implementation.

| Memetic Algorithms With GRASP Initial Solutions | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Circuit** | **GA-GR** | | **GA-GR-II** | | **GA-GR-FI** | | **GA-GR-IFI** | | **GA-GR-DHC** | |
| | Cuts | Time | Cuts | Time | Cuts | Time | Cuts | Time | Cuts | Time |
| Fract | 49 | 0:01 | 43 | 0:01 | 35 | 0:02 | 34 | 0:02 | 28 | 0:01 |
| Prim1 | 133 | 0:08 | 125 | 0:08 | 125 | 0:10 | 113 | 0:10 | 120 | 0:10 |
| Struct | 167 | 0:20 | 178 | 0:21 | 164 | 0:25 | 151 | 0:25 | 132 | 0:26 |
| Prim2 | 369 | 0:33 | 387 | 0:35 | 358 | 0:42 | 369 | 0:42 | 339 | 0:42 |
| Ind1 | 130 | 0:25 | 114 | 0:26 | 143 | 0:31 | 102 | 0:32 | 92 | 0:31 |
| Bio | 324 | 1:17 | 328 | 1:18 | 242 | 1:32 | 275 | 1:32 | 303 | 1:40 |
| Ind2 | 866 | 3:23 | 740 | 3:32 | 761 | 4:16 | 648 | 4:23 | 801 | 4:21 |
| Ind3 | 1551 | 5:01 | 1513 | 5:27 | 1762 | 6:20 | 1444 | 7:08 | 1345 | 6:36 |
| Avq.s | 997 | 6:13 | 992 | 6:40 | 1059 | 8:58 | 947 | 8:04 | 986 | 7:55 |
| Avq.l | 1028 | 7:12 | 996 | 7:50 | 983 | 10:31 | 1001 | 9:21 | 988 | 9:13 |
| **Total** | 5614 | 1473 | 5416 | 1578 | 5632 | 2007 | 5084 | 1939 | 5134 | 1895 |
| **%Imp** | +12% | -18% | +15% | -23% | +11% | -39% | +20% | -37% | +19% | -36% |

Table 4: Performance of 4-Way GA/MA Partitioning (GRASP Initial)

## 4.3 Genetic-based Placement Algorithm

In this section, we present a Genetic Algorithm for standard cell placement. In this algorithm, a solution string is represented by a set of alleles (the number of alleles equal to the number of cells). Each allele indicates the index, the x- coordinates and row number of the cell. Figure 15a illustrates the string encoding of the cell placement given in Figure 15b. Figure 16 shows a pseudo-code of the Genetic-based placement



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| cell_index | 2 | 3 | 1 | 8 | 7 | 6 | 5 | 4 |
| row_number | 0 | 3 | 0 | 2 | 1 | 2 | 3 | 1 |
| x−coordinate | 0 | 20 | 50 | 30 | 0 | 50 | 40 | 30 |

Figure 15: String Encoding

algorithm. The fitness function used is the reciprocal of the total *HPWL*[6] for all the nets.

$$F = \frac{1}{\frac{1}{2}\sum_{i=1}^{n} HPWL_i} \tag{4}$$

[6]HPWL is the Half Perimeter Wire Length.

```
1. Encode Solution Space for Placement
2. set popsize, max_gen, gen=0;
   set crossover_rate, mutation_rate;
3. Generate initial population randomly
4. **While** (gen $\leq$ max_gen)
      **For** (i=1 to popsize/2)
         Select_parents(mate1,mate2);
         if (random(0,1) $\leq$ crossover_rate)
            child = Do_Crossover(mate1,mate2);
         if (random(0,1) $\leq$ mutation_rate)
            Mutation(offspring) and evaluate offspring;
      **End For**
      Replacement();
      gen = gen + 1 ;
   **End While**
5. Return best solution in current population.
```

Figure 16: A Genetic Placement Algorithm

where *HPWL* is the sum of the half perimeter of the smallest bounding rectangle for each net as illustrated in Figure 17. $HPWL_i$ is the estimate wire length of net *i* and

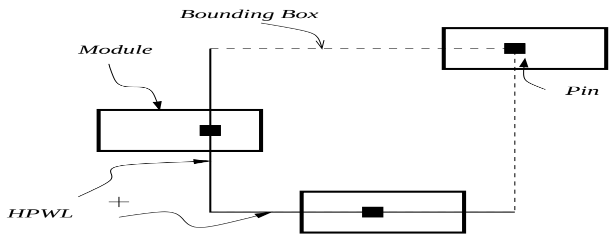


Figure 17: Half Perimeter Wire Length (HPWL)

n is the number of nets. In the implementation, cell overlaps are removed and row lengths are adjusted before evaluating the chromosome. Removing the overlaps after every generation not only gives the algorithm a more accurate picture of the wire length but also gives the algorithm repeated chances to optimize the circuit after it has been perturbed by overlap removal (YA02). For the initial population construction, some placement solutions produced by the Cluster-Seed method are injected to increase the convergence rate. The traditional crossover operator used in GAs may produce infeasible solutions for the standard cell placement problem, therefore a crossover operator called *Order Crossover* is considered as shown in Figure 18. Following crossover, each offspring is mutated with a probability equal to the mutation rate. In GAs, mutation produces incremental random changes in the offspring generated through crossover. It not only plays the crucial role of replacing the gene values lost during the selection process, but also provides the gene values that were not presented in the initial population. Two mutation operators $m_1$ and $m_2$ were tested. Operator $m_1$ mutates an individual by interchanging randomly selected pair of cells without changing the x-coordinate and row number. Figure 19 illustrates the mutation process. Its random nature allows for a broader exploration of the solution space. However, it typically increases a string's score due to its disruptive effect on the placement solution. Therefore, another muta-

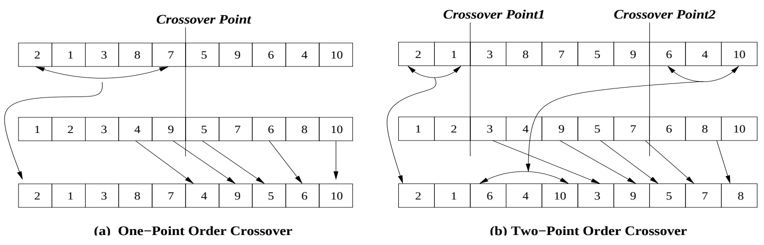


Figure 18: Order Crossover for Circuit Placement

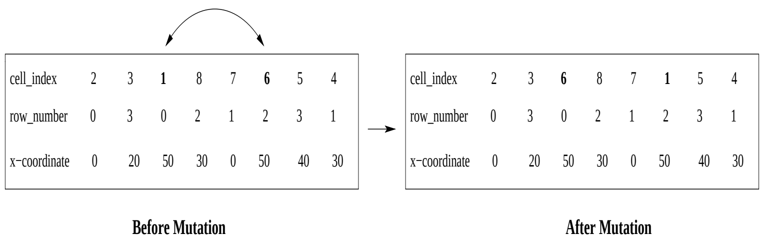

cell_index 2 3 1 8 7 6 5 4
row_number 0 3 0 2 1 2 3 1
x−coordinate 0 20 50 30 0 50 40 30
cell_index 2 3 6 8 7 1 5 4
row_number 0 3 0 2 1 2 3 1
x−coordinate 0 20 50 30 0 50 40 30
Before Mutation
After Mutation

Figure 19: Placement Mutation Operator

tion operator $m_2$ was considered, where a cell $c_1$ is randomly chosen and its location is swapped with another randomly selected cell $c_2$, if and only if cell $c_2$ is located in the same cell, or up or down one row from cell $c_1$.

The replacement process is done by replacing the two worst individuals in the population with the offsprings only if the latter are better than the worst two individuals.

## 4.4 Memetic Algorithm for Circuit Placement

The proposed Memetic Algorithm (shown in Figure 20) for circuit placement is based on the Genetic Algorithm introduced earlier. In each generation, a Tile-based local search heuristic is performed on part of the population to improve their fitness. The number of generations used to terminate the algorithm is set to 10.

For the circuit placement problem, the pure GA is combined with Tile-based local search in three different ways, referred to as performing local search on part of the population: (i) before the crossover "GA-ME-1" (ii) after the crossover "GA-ME-2" (iii) before and after the crossover "GA-ME-3".

The first column in Table 5 presents results produced by the pure Genetic Algorithm. For all the results the population size is 14 and the generation size is 10. Cluster-Seed (presented in Section 2.2.2) based results are injected into both pure GA and Memetic algorithms as part of the initial population. From the table, it can be seen that the amount of improvement achieved is 44%, 43% and 47% respectively. Obviously, integrating GA with local search in the first two methods reduces the amount of wire-length and CPU time on average by 45% and 17% respectively. The last approach enhances the wire-length quality at the expense of an increase in CPU time on average by 40%.

```
1. Encode Solution Space for Placement
2. set popsize, max_gen, gen=0;
   set crossover_rate, mutation_rate;
3. Generate initial population randomly
4. Evaluate the initial population
5. **While** (gen $\leq$ max_gen)
     Apply GA
       Apply Tile-based algorithm to Population;
   **End While**
6. Return best solution in current population.
```

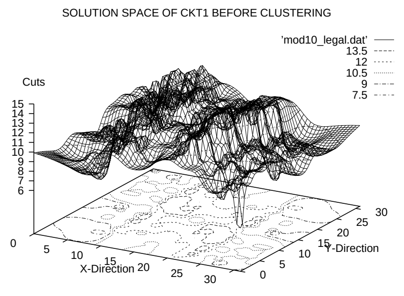
Figure 20: A Memetic Placement Algorithm

| **Performance of Memetic Algorithms For Placement** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Circuit** | **Pure GA** | | **GA-ME-1** | | **GA-ME-2** | | **GA-ME-3** | |
| | X+Y | Time | X+Y | Time | X+Y | Time | X+Y | Time |
| Fract | 61207 | 3.5 | 34997 | 15.1 | 35942 | 15.4 | **34953** | 31.1 |
| Prim1 | 1.78e+06 | 29.6 | **995713** | 233.2 | 1.02e+06 | 219.6 | 1.00e+06 | 467.8 |
| Struct | 930148 | 114.0 | 504720 | 288.8 | 490276 | 251.0 | **460289** | 600.7 |
| Ind1 | 4.42e+06 | 185.4 | 2.21e+06 | 926.9 | **2.07e+06** | 876.6 | 2.14e+06 | 1806.0 |
| Prim2 | 1.03e+07 | 263.4 | 5.48e+06 | 741.3 | **5.44e+06** | 685.8 | 5.47e+06 | 1451.5 |
| Bio | 7.44e+06 | 863.5 | 3.98e+06 | 1256.8 | 4.01e+06 | 1240.0 | **3.82e+06** | 2492.7 |
| Ind2 | 6.39e+07 | 2933.3 | 2.97e+07 | 3185.7 | 3.08e+07 | 3055.8 | **2.78e+07** | 6522.6 |
| Ind3 | 1.58e+08 | 4688.6 | 9.92e+07 | 5716.4 | 1.01e+08 | 5313.4 | **9.31e+07** | 11405.4 |
| avq.small | 3.80e+07 | 7697.7 | 2.10e+07 | 5278.9 | 2.09e+07 | 5033.9 | **2.02e+07** | 10130.6 |
| avq.large | 5.08e+07 | 11699.4 | 2.34e+07 | 6422.6 | 2.35e+07 | 6065 | **2.27e+07** | 12387.4 |
| **Total** | 33.55+07 | 28475 | 18.59+07 | 24065 | 18.87+07 | 22753 | 17.67+07 | 47295 |
| **% Imp** | 0% | 0% | +44% | +15% | +43% | +20% | +47% | -39% |

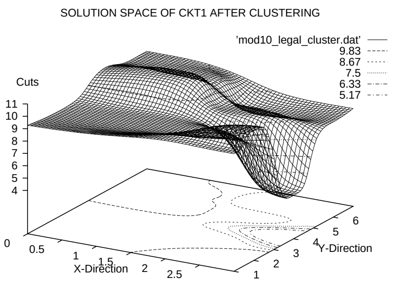
Table 5: Results of Memetic Algorithms for Circuit Placement

## 5 Hierarchical Approach

Clustering usually serves as a bottom-up preprocessing stage in a hierarchical partitioning and placement environment. A good clustering method should identify groups of cells which will eventually end up together in the final partitioning and placement stages. Figure 21 shows the effect of clustering in reducing the complexity of a simple circuit with 10 modules and 15 nets. It is clear that the number of local minima is reduced after clustering the original circuit. This process enables the Genetic Algorithm to explore the solution space more effectively and to converge to a good neighborhood solution in a short period of time.



(a)



(b)

Figure 21: **3D View of Circuit Before/After Clustering**

### 5.1 Hierarchical Approach for Circuit Partitioning

The hierarchical based partitioning algorithm uses a modified Memetic Algorithm that integrates a Genetic Algorithm, GRASP, local search and clustering. Initially the system starts by calculating statistical information and the attributes of the circuit. This information is then used to collapse a set of nodes to form a single component (i.e super nodes). The algorithm then proceeds with an encoding and initialization phase during which each string in the population is assigned a uniformly distributed random point in the solution space. Following clustering, a $\delta$% of the population is injected with good initial solutions based on a constructive technique (i.e GRASP). In the final stage of the Memetic Algorithm a local search heuristic is used on the flattened network to further optimize local partitions of cells. The combined clustering and local search methodology can be viewed as a combined bottom-up and top-down approach.

Tables 6, 7 show the results obtained with random initial solutions. The first column (GA-FLAT) of each table are based on applying a pure Genetic Algorithm to a flat circuit (i.e no clustering involved). The results in the second column (GA-GC-FI) are based on a Memetic Algorithm with simple local search applied at the end of the last two generations. The technique (GA-GC-FI) involves clustering the circuit to a single level and then applying the Memetic Algorithm. The third column (GA-GC-IFI) is similar to (GA-GC-FI) except that local search is applied at the end of each generation to all individuals in addition to a local search to the best chromosome. The final column (GA-GC-DHC) is based on a dynamic hill climbing algorithm. The last two rows in the tables present the total cut-size and amount of improvement achieved with respect to the pure Genetic Algorithm implementation. It is clear from Tables 6, 7 that the best results achieved are those based on both clustering and simple local search. The amount of improvement in cut-size achieved using (GA-GC-FI) are 13% for two way partitioning and 25% for four-way partitioning respectively. The amount of improvement using (GA-GC-IFI) is 16% for two-way partitioning and 22% for two way partitioning. A clustering Memetic Algorithm (GA-GC-DHC) with dynamic hill climbing capability improves the solution quality by 24% for two-way partitioning and 35% for four-way partitioning. The computation time involved using all techniques improves upon those obtained using a pure GA on the flat circuit.

| **Random Clustered Based Memetic Techniques** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Circuit** | **GA-FLAT** | | **GA-GC-FI** | | **GA-GC-IFI** | | **GA-GC-DHC** | |
| | Cuts | Time | Cuts | Time | Cuts | Time | Cuts | Time |
| Fract | 46 | 0:01 | 11 | 0:01 | 11 | 0:01 | 11 | 0:01 |
| Prim1 | 87 | 0:06 | 90 | 0:04 | 68 | 0:04 | 65 | 0:04 |
| Struct | 77 | 0:13 | 49 | 0:08 | 49 | 0:09 | 49 | 0:09 |
| Prim2 | 196 | 0:24 | 168 | 0:17 | 168 | 0:19 | 155 | 0:23 |
| Ind1 | 78 | 0:17 | 67 | 0:13 | 67 | 0:14 | 62 | 0:16 |
| Bio | 211 | 0:46 | 133 | 0:47 | 98 | 0:47 | 92 | 0:48 |
| Ind2 | 299 | 2:09 | 351 | 1:55 | 327 | 2:08 | 301 | 2:16 |
| Ind3 | 555 | 3:15 | 434 | 3:07 | 404 | 3:25 | 358 | 3:33 |
| Avq.s | 511 | 3:35 | 462 | 3:41 | 467 | 3:51 | 435 | 3:58 |
| Avq.l | 486 | 4:08 | 453 | 3:45 | 485 | 3:15 | 407 | 3:12 |
| **Total** | 2546 | 894 | 2218 | 791 | 2144 | 853 | 1935 | 880 |
| **%Imp** | 0% | 0% | +13% | +12% | +16% | +4% | +24% | +2% |

Table 6: 2-Way Clustered Based Partitioning

| **Random Clustered Based Memetic Techniques** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Circuit** | **GA-FLAT** | | **GA-GC-FI** | | **GA-GC-IFI** | | **GA-GC-DHC** | |
| | Cuts | Time | Cuts | Time | Cuts | Time | Cuts | Time |
| Fract | 67 | 0:01 | 36 | 0:01 | 37 | 0:01 | 35 | 0:01 |
| Prim1 | 162 | 0:07 | 114 | 0:05 | 114 | 0:05 | 112 | 0:06 |
| Struct | 253 | 0:18 | 127 | 0:07 | 135 | 0:08 | 124 | 0:09 |
| Prim2 | 447 | 0:29 | 347 | 0:24 | 278 | 0:26 | 275 | 0:34 |
| Ind1 | 140 | 0:21 | 90 | 0:14 | 94 | 0:15 | 94 | 0:14 |
| Bio | 285 | 1:03 | 234 | 0:46 | 255 | 0:53 | 230 | 0:58 |
| Ind2 | 1287 | 2:55 | 862 | 2:35 | 859 | 2:50 | 754 | 3:03 |
| Ind3 | 1581 | 4:18 | 1134 | 3:24 | 1131 | 3:47 | 921 | 3:55 |
| Avq.s | 1079 | 4:57 | 894 | 3:53 | 857 | 4:05 | 749 | 4:07 |
| Avq.l | 1071 | 5:38 | 963 | 4:03 | 916 | 4:37 | 847 | 4:45 |
| **Total** | 6372 | 1207 | 4801 | 932 | 4676 | 1027 | 4141 | 1072 |
| **%Imp** | 0% | 0% | +25% | +22% | +27% | +15% | +35% | +12% |

Table 7: 4-Way Clustered Based Partitioning

### 5.2 Hierarchical Approach for Standard Cell Placement

Early methods of clustering performed the desired circuit size reduction in a single level (e.g. (MG89)). Research has recently shown that clustering in steps (illustrated in Figure 22), reducing the circuit size gradually by adding intermediate levels to the hierarchy, produces superior results by permitting more gradual de-clustering (KAKS97). This gradual clustering is often called "multi-level" or "hierarchical" clustering. During de-clustering in a single clustering level heuristic, the difference between positions in clustered cells and flat circuit cells can be substantial, and significant iterative improvement is necessary to achieve a high quality solution. In a multi-level heuristic, much smaller differences are created between levels of the hierarchy, because it is built slowly. During de-clustering, these differences are more easily managed by simple interchange heuristics, resulting in a superior quality solution in a shorter amount of time (AMA01).

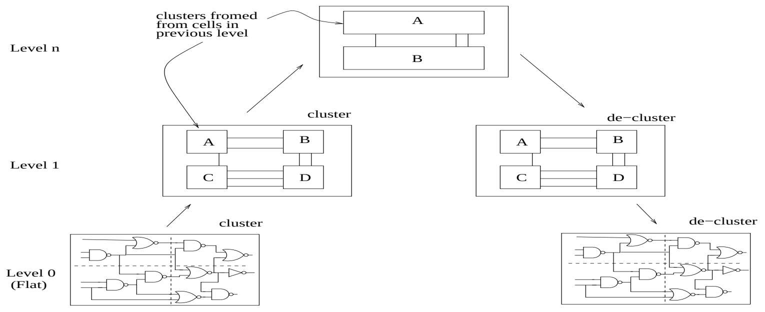


Figure 22: Multilevel Clustering Hierarchy.

In this paper, a simple multi-level clustering technique called "Weighted Hyperedge Clustering" (ATV01), (which is an extension to Karypis et. al work (KAKS97)) is used for the hierarchical placement approach. As seen in the pseudo-code in Figure 23, an upper and lower width limit is determined based on the cell widths in the current hierarchical level. As a potential clustering of cells is examined, a new cluster is only created if the sum of the constituent cells' widths is in the current hierarchical level. As a potential clustering of cells is examined, a new cluster is only created if the sum of the constituent cells' widths is between these width limits. This limitation on sizes prevents excessively large clusters from impeding improvement, yet still reduces the problem size.

Table 8 shows the results obtained by a Genetic Algorithm (Pure GA), a GA with a clustering technique (GA-GC), a Memetic Algorithm with a simple Tile-based local search (GA-Tile) and a GA based on a clustering technique with Tile-based local search embedded (GA-GC-Tile). It is clear from this table that the algorithm based on both clustering and simple local search produces high quality solutions. The results shown in the first two columns indicate that by combining the clustering technique with GA the computation time was largely reduced by 85% but the quality of the solution deteriorated compared to results obtained for flat benchmarks. The amount of improvement in total estimated wire-length achieved using Memetic algorithm (GA-Tile) are 47% but the computation time involved using a Memetic algorithm increases largely. A clustering Memetic Algorithm (GA-GC-Tile) improves the solution quality by 54% and the computation time is less than that of the flat Memetic Algorithm (GA-Tile).

```
1. Sort nets by increasing size
2. For each sorted net
     If  no cell on net is clustered
          If  sum of cell widths on net is within limits
               Cluster all cells on net
          End If
     End If
   End For
3. For each sorted net
     If  sum of unclust cell widths is within limits
          Cluster all unclust cells on net
     End If
   End For
4. For each cell in circuit
     If  not clustered
          Create a new cluster from cell
     End If
   End For
```

Figure 23: Weighted Hyperedge Clustering

| Performance of Hierarchical Approach | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Circuit** | **Pure GA** | | **GA-GC** | | **GA-Tile** | | **GA-GC-Tile** | |
| | X+Y | Time | X+Y | Time | X+Y | Time | X+Y | Time |
| Fract | 61207 | 3.5 | 54548 | 1.5 | 34953 | 31.1 | **34627** | 22.9 |
| Prim1 | 1.78e+06 | 29.6 | 2.18e+06 | 9.4 | 1.00e+06 | 467.8 | **964815** | 438.2 |
| Struct | 930148 | 114.0 | 1.39e+06 | 14.3 | **460289** | 600.7 | 464454 | 544.8 |
| Ind1 | 4.42e+06 | 185.4 | 3.87e+06 | 47.6 | 2.14e+06 | 1806.0 | **1.97e+06** | 1318.4 |
| Prim2 | 1.03e+07 | 263.4 | 1.21e+07 | 54.2 | 5.47e+06 | 1451.5 | **5.31e+06** | 1286.9 |
| Bio | 7.44e+06 | 863.5 | 6.15e+06 | 64.6 | 3.82e+06 | 2492.7 | **2.77e+06** | 2157.9 |
| Ind2 | 6.39e+07 | 2933.3 | 7.42e+07 | 642.5 | **2.78e+07** | 6522.6 | 3.10e+07 | 5420.7 |
| Ind3 | 1.58e+08 | 4688.6 | 2.86e+08 | 816.3 | 9.31e+07 | 11405.4 | **8.06e+07** | 7521.4 |
| avq.small | 3.80e+07 | 7697.7 | 4.51e+07 | 1203.2 | 2.02e+07 | 10130.6 | **1.35e+07** | 9711.6 |
| avq.large | 5.08e+07 | 11699.4 | 6.13e+07 | 1606.4 | 2.27e+07 | 12387.4 | **1.60e+07** | 11060.4 |
| **Total** | 33.46+07 | 28478 | 49.17+07 | 4460 | 17.67+07 | 47294 | 15.26+07 | 39478 |
| **% Imp** | 0% | 0% | -32% | 85% | 47% | -39% | 54% | -27% |

Table 8: Hierarchical Clustering for Circuit Placement

## 6 Conclusions and Future Work

This paper presented several approaches to integrating Evolutionary Computation models with local search techniques (i.e Memetic Algorithms) for efficiently solving underlying VLSI circuit partitioning and placement problems. The methodology presented in the paper explained how clustering reduces the complexity of the circuit and thereby enables the Evolutionary technique to explore the solution space more effectively. Constructive heuristic techniques in the form of GRASP and Cluster Growth were utilized to inject the initial population with good initial solutions to diversify the search and exploit the solution space. Furthermore, the local search technique was able to enhance the convergence rate of the Evolutionary Algorithm by finely tuning the search on the immediate area of the landscape being considered.

For the partitioning problem, Memetic Algorithms based on GRASP, local search and clustering achieve on average an improvement of 35% over a traditional Genetic Algorithm. Whereas for the circuit placement, Memetic Algorithms based on Cluster-Growth, Tile-based local search and hierarchical clustering reduce the total estimated wire-length on average by 54%. This clearly indicates that Memetic Algorithms are a powerful algorithmic paradigm for evolutionary computing. Future work involves using more efficient ways to combine a local search heuristic with GA to produce high quality solutions in less time, especially for the circuit placement problem.

## References

S. Areibi, M. Moussa, and H. Abdullah. A Comparison of Genetic/Memetic Algorithms and Other Heuristic Search Techniques. In *International Conference on Artificial Intelligence*, pages 660–666, Las Vegas, Nevada, June 2001.

S. Areibi. An Integrated Genetic Algorithm With Dynamic Hill Climbing for VLSI Circuit Partitioning. In *GECCO 2000*, pages 97–102, Las Vegas, Nevada, July 2000. IEEE.

S. Areibi, M. Thompson, and A. Vannelli. A Clustering Utility Based Approach for ASIC Design. In *14th Annual IEEE International ASIC/SOC Conference*, pages 248–252, Washington, DC, September 2001. IEEE, ACM.

S. Areibi and A. Vannelli. An Efficient Clustering Technique for Circuit Partitioning. In *IEEE ISCAS*, pages 671–674, San Diego, California, 1996.

S. Dutt and W. Deng. VLSI Circuit Partitioning by Cluster-Removal Using Iterative Improvement Techniques. In *IEEE International Conference on CAD*, pages 194–200. ACM/IEEE, 1996.

S. Dutt and H. Theny. Partitioning Around Roadblocks: Tackling Constraints with Intermediate Relaxations. In *IEEE International Conference on CAD*, pages 350–355, November 1997.

C.M. Fiduccia and R.M. Mattheyses. A Linear-Time Heuristic for Improving Network Partitions. In *Proceedings of 19th DAC*, pages 175–181, Las Vegas, Nevada, June 1982. ACM/IEEE.

T. Feo, M. Resende, and S. Smith. A Greedy Randomized Adaptive Search Procedure for The Maximum Ind ependent Set. *Operations Research*, 1994. Journal of Operations Research.

M.R. Garey and D.S. Johnson. *Computers and Intractability*. Freeman, San Francisco CA, 1979.

D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Company, Inc, Reading, Massachusetts, 1989.

L. Hagen and A.B. Kahng. A New Approach to Effective Circuit Clustering. In *IEEE International Conference on CAD*, pages 422–427, 1992.

G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel Hypergraph Partioning: Application in VLSI Design. In *Proceedings of* 35*th DAC*, pages 526–529, Las Vegas, Nevada, June 1997. ACM/IEEE.

B.W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell System Technical Journal*, 49(2):291–307, February 1970.

Sung-Mo Kang and Yusuf Leblebici. *CMOS Digital Integrated Circuits*. McGraw-Hill Publishing Company, Inc, 2003.

P.G. Karger and B.T. Preas. Automatic Placement: A Review of Current Techniques. In *Proceedings of The* 23*rd DAC*, pages 622–629, Las Vegas, Nevada, 1986. IEEE/ACM.

S. Mallela and L.K. Grover. Clustering Based Simulated Annealing for Standard Cell Placement. In *Proceedings of The* 26*th DAC*, pages 312–317, Las Vegas, Nevada, 1989. IEEE/ACM.

Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlog, Berlin, Heidelberg, 1992.

J. Rabaey, A. Chandrakasan, and B. Nikolic. *DIGITAL INTEGRATED CIRCUITS*. Pearson Education Publishing Company, Inc, 2003.

B.M. Riess, K. Doll, and F.M Johannes. Partitioning very large circuits using analytical placement techniques. In *Proceedings of* 31*st DAC*, pages 646–651, Las Vegas, Nevada, 1994. ACM/IEEE.

K. Roberts and B. Preas. Physical Design Workshop 1987. Technical report, MCNC, Marriott's Hilton Head Resort,South Carolina, April 1987.

L.A. Sanchis. Multiple-Way Network Partitioning. *IEEE Transactions on Computers*, 38(1):62–81, January 1989.

C. Sechen and D. Chen. An improved Objective Function for Min-Cut Circuit Partitioning. In *Proceedings of ICCAD*, pages 502–505, San Jose, California, 1988.

K. Shahookar and P. Mazumder. VLSI Cell Placement Techniques. *ACM Computing Surveys*, 23(2):143–220, 1991.

Z. Yang and S. Areibi. Global Placement Techniques for VLSI Physical Design Automation. In *15th International Conference on Computer Applications in Industry and Engineering*, pages 243–247, San Diego, California, November 2002. ISCA.